Пояснювальна записка Казаріна Максима Вікторовича

до дипломного проєкту

на тему: «Автоматизована система контролю

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Казаріну Максиму Вікторовичу

1. Тема проєкту «Автоматизована система контролю відвідування зборів», керівник проєкту старший викладач Тимофєєва Юлія Сергіївна, затверджені наказом по університету від «_7_»_травня_ 2020 р. №_1081-с___

2. Термін подання студентом проєкту 9 червня 2020р._____

3. Вихідні дані до проєкту розробка під мобільну операційну систему Android, доступність автентифікації за допомогою хмарного сервісу Firebase, підтримка протоколів безпровідного зв’язку.

4. Зміст пояснювальної записки опис предметної області, аналіз готових рішень, формування вимог до мобільного застосунку, вибір технологій розробки, розробка мобільного застосунку, тестування системи, керівництво користувача, висновки, використані джерела.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо) Діаграма класів, Діаграма використання, Діаграма послідовності, Діаграма діяльності.

6. Дата видачі завдання 17 лютого 2020р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Реалізація підтримки стандарту 802.11	10 квітня 2020 року	
2	Аналіз існуючих рішень	17 квітня 2020 року	
3	Розробка серверу для застосунку	23 квітня 2020 року	
4	Інтеграція авторизації користувача	29 квітня 2020 року	
5	Інтеграція автентифікації користувача	5 травня 2020 року	
6	Оформлення пояснювальної записки	11 травня 2020 року	
7	Розробка та опис графічного матеріалу	15 травня 2020 року	

Студент Максим КАЗАРІН

Керівник Юлія ТИМОФЄЄВА

АНОТАЦІЯ

Казарін М.В. Автоматизована система контролю відвідування заходів. КПІ імені Ігоря Сікорського, Київ, 2020.

У дипломному проєкті розроблена система для пришвидшення процесу проходження контролю відвідування заходів. Сформований опис предметної області. Був проведений аналіз готових рішень та обґрунтований вибір основних технологій, сервісів та платформ розробки. Сформоване завдання та критерії до системи. Описана функціональна частина мобільного застосунку, а саме: клієнт-серверна архітектура, база даних, клієнтський інтерфейс та мережева передача даних. Розроблені діаграми використання, класів, діяльності, послідовності та наданий їх опис. Показані основні етапи тестування системи.

Робота містить 61 сторінки пояснювальної записки, 20 рисунків, 1 таблицю, 19 літературних джерел та 5 додатків.

ANNOTATION

Kazarin M.V. Automated event attendance control system. Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 2020.

In the diploma project the system to speed up the process of attending events is developed. The description of the subject area is formed. The analysis of ready decisions was carried out and the choice of the basic technologies, services and development platforms was substantiated. The task and criteria to the system are formed. The functional part of the mobile application is described, namely: client-server architecture, database and client interface and network data transmission. Diagrams of use-case, classes, activities, sequences are developed and their description is given. The main stages of system testing are shown.

The work contains 61 pages of explanatory note, 20 figures, 1 table, 19 references and 5 appendices.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

UHF — Ultra High Frequency

NFC — Near Field Communication

ПК — Персональний Комп’ютер

BaaS — Backend as a Service

JVM — Java Virtual Machine

IP — Internet Protocol

API — Application Public Interface

DoS — Denial of Service

UUID — Unique Uniform Identifier

REST — Representational State Transfer

ВСТУП

Сучасна розробка програмного забезпечення направлена на автоматизацію та поліпшення життя людей. З'являється все більше нових додатків, які дозволяють звільнити себе від буденних обов'язків. Програмні рішення вже проникли у різні сфери діяльності людини, рівень автоматизації є прямим показником ефективності, надійності та оптимальності.

Якщо взяти до уваги такий процес як контроль відвідування. Він ще з початку свого зародження виконувався в писемному вигляді. Історія мала випадки автоматизації цього процесу за допомогою табелів обліку робочого часу та турнікети, останні ще й досі користуються популярністю. Однак ще є місця, де використовують самий перший варіант контролю відвідування. І це не дивно, бо саме він дозволяє мати чітке представлення про кількість присутніх, та тих, хто запізнився. Це невіддільна частина статистики підприємства, закладу чи заходу. За її допомогою, після детального аналізу, формуються рішення спрямовані на редагування часових проміжків для поліпшення кількості відвідувачів, зменшення тих, хто запізнюється.

В сучасному світі будь-який захід не обходиться без контролю відвідувачів. А цей процес потребує великої кількості людських ресурсів тому автоматизовані системи, які допоможуть покращити якість та швидкість надання таких послуг є достатньо важливими.

Метою проєкту є пришвидшення процесу проходження контролю відвідувань заходів, забезпечення незалежності їх від місця проведення та надати користувачам рівні права та функціональність. І також звільнити організаторів від довготривало процесу створення заходів заради зосередження їх над самим заходом.

В даній роботі описується розробка функціональної частини мобільного застосунку для розв'язання задач зручного та швидкого контролю відвідування зборів. Під потреби реалізації була обрана мова програмування Java. Розглянуті

методи підтримки двох контрактів: перший — користувач одного має змогу відмітитись в одному заході, другий — власник заходу не може бути його учасником. Описані способи автентифікації користувача у застосунку та використані при цьому сервіси. Проєктування структури бази даних та наочної роботи з нею. Структуровано модель обміну даними в межах локальної безпровідної мережі. Розглянута топологія системи. Описані способи сповіщення користувачів через локальну мережу та хмарний сервіс. Представлений метод накладання додаткових обмежень на основі сучасного двовимірного штрих-коду. Було розроблено наступні кресленики: діаграма варіантів використання, діаграма послідовності, діаграма класів та діаграма діяльності.

Об’єкт дослідження — система контролю.

Предмет дослідження — автоматизована система контролю відвідування заходів.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

В даному дипломному проєкті виконується створення спеціального програмного забезпечення для розв’язання основних задач контролю відвідування. В першу чергу слід визначити термін відвідування.

Відвідування — це процес, який обумовлює присутність особи у заздалегідь зазначеному місці, зазвичай на невизначений час, але характеризується постійністю та періодичністю.

Для забезпечення нормативності та систематичності були розроблені системи контролю.

Система контролю[1] — сукупність програмно-апаратних технічних засобів контролю і засобів управління, що мають на меті обмеження і реєстрацію входу-виходу об'єктів (людей, транспорту) на заданій території через «точки пропуску». Бувають двох типів: мережеві та автономні.

Автономні — досить дешеві системи, які не мають сполучення з комп’ютером, і виконують контролю внутрішніми засобами, забезпечені додатковим живленням і водостійким захисним корпусом.

Переваги:

— ціна;

— надійність.

Недоліки:

— слабка гнучкість;

— відсутність віддаленого контролю;

— відсутність звітності.

Мережеві — системи для великих територій, де кожний контролер з’єднаний з комп’ютером, де відбувається централізоване управління. Можливе використання безпровідних технологій таких як Bluetooth, Wi-Fi, ZigBee, GSM.

Переваги:

— наявність звітності;

— швидке оновлення даних;

— площа покриття;

— легка інтеграція нових систем.

Недоліки:

— ціна;

— обслуговування.

Якщо розглядати програмне забезпечення відвідування заходів, то воно є різновидністю мережевої системи контролю, і ввібрало в себе всі її переваги. Повністю базується на бездротових технологія передачі даних, тому вся система складається з «Check-in» точок пристроїв адміністраторів та серверу. Грамотне розташування всього цього обладнання з розділенням на входи і виходи здатне покривати великі площі розмірами зі стадіон. Достатнє розповсюдження глобальної мережі Інтерент та наявність віддаленого сервера дозволяє зберігати дані та формувати звітність з можливим наступним його аналізом. Деякі системи контролю відвідування потребують функціоналу оповіщення організаторів про прибуття особливих гостей, що накладає нові потреби в реалізації методів внутрішнього зв’язку. А оскільки всі мережеві системи мають підтримку безпровідного зв’язку, то рішенням цього виступає спілкування через локальну мережу. Іншим варіантом інтеграції виступає платіжна система, яка дозволяє розраховуватись напряму через систему контролю, і яка набула популярності між благодійних заходів. Ключовим елементом систем контролю відвідування заходів виступає процедура автентифікації, що перекладає обов’язки з підтвердження особистості з адміністратора заходу на на систему.

Автентифікація[2] — процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора з наступним наданням доступу до роботи з інформаційною системою.

Способи автентифікації користувача в системі контролю відвідування:

— логін та пароль, що є одним з найстаріших способів, і що продовжує використовуватись;

— квиток на пошту, такий спосіб потребує наявність квитка у користувача для доступу до заходу;

— QR-Code, що може бути як компактною версією квитка так і виступає для генерації запрошення на пристрої користувача після сканування.

— відбитки пальців, біометричний тип автентифікації що потребує додаткових даних від користувача під час реєстрації, але вважається більш надійним;

— сканування обличчя, наступний біометричний тип автентифікації, теж потребує додаткових даних від користувача під час реєстрації, і теж вважається більш надійним;

— UHF/NFC, безконтактний спосіб, що супроводжується автентифікацією через персональний пристрій чи годинник користувача.

Дана предметна область описує вузьку область використання систем контролю відвідування. І показу в якому напрямку розвиваються підходи використання рішень у сучасному світі. Було визначено ключові аспекти задач та цілей системи, та як використовуються новітні розробки для швидшого досягнення мети систем. Також було визначено які висновки можна робити по статистичним даним функціонуючих систем. Описані цілі автентифікації у системах контролю, та можливі її методи.

2 АНАЛІЗ ГОТОВИХ РІШЕНЬ

Існує досить багато аналогів автоматизованих систем для перевірки відвідувачів заходів. Кожна з них розрахована на різну кількість користувачів, площу, яку охоплює, використовує різні пристрої які відповідають за контроль.

2.1 Програмне рішення «OnArrival by Cvent»

«OnArrival by Cvent»[3] — це система розроблена компанією Cvent для організації заходів на великих площах та для великої кількості людей. Вона розрахована на декілька контрольних точок, розташованих при входах. Ці контрольні точки не потребують асистентів, кожен гість самостійно проходить авторизацію у системі і отримує свій значок. Система має свою платіжну систему за допомогою якої користувачі можуть робити благодійні внески та розплачуватись за аксесуари.

Найкраще для планувальників подій, яким потрібне комплексне рішення, є також синхронізація зі своєю платформою управління подіями для більшої ефективності та точності.

Переваги:

— усі дані про гостя надійно та автоматично синхронізуються через платформу управління подіями Cvent;

— гостям доступний "режим кіоску", що надає друк значків на вимогу та навіть обробку платежів;

— наявна система оповіщення прибуття важливих гостей;

— відстеження відвідування сеансу за допомогою портативних ІК-сканерів QR коду;

— зазначення власних інтересів кожним учасником заходу;

— можливість перевірити в учасників заходу, а також будь-які додаткові сеанси під час події;

— повна функціональність онлайн або офлайн;

— кілька співробітників можуть керувати реєстрацією з декількох точок входу;

— фільтрація та пошук в списку відвідувачів по імені.

Недоліки:

— система потребує додаткової техніки, такої як принтери значків, портативні сканери, сканерами банківських карток для кіоску;

— складний процес налаштування заходу;

— немає способу налаштування розміщення відвідувачів по території.

2.2 Програмне рішення «Social Tables Check-in»

«Social Tables Check-in»[4] — це рішення від Social Tables яке являє собою мобільний застосунок на смартфон та комп’ютер. Він встановлюється та використовується на персональних пристроях. Користувач може створити власний захід та запросити на нього друзів зі свого списку контактів. Додаток має список публічних заходів, які створені іншими користувачами, та сортування їх за датами та місцями.

Найкраще для менеджерів заходів, яким потрібно створювати заходи без додаткового обладнання та з мінімальним налаштуванням системи.

Переваги:

— наявна зручна система фільтрів;

— доступна версія для персонального комп’ютера;

— можливо перевірити статус кожного гостя;

— керування подіями з будь-якої точки світу, використовуючи лише мобільні пристрої;

— оснащена функціями тестування параметрів заходів;

— можливо отримати інформацію про потреби свого клієнта відносно свої події, щоб отримати цінну інформацію про відвідувачів;

— доступна організація розміщення внутрішніх елементів;

— має впроваджений список сервісів постачальників, і налагоджену роботу в одній системі.

Недоліки:

— старий дизайн програми, який погано працює на нових версіях планшетів;

— постійна зміна публічного програмного інтерфейсу від версії до версії;

— зображення квитка в форматі PDF робить його складним для зчитування;

— складність налаштування, та перевантаженість користувацького інтерфейсу.

2.3 Програмне рішення «Boomset»

«Boomset»[5] — це черговий мобільний застосунок розроблений компанією Boomset. Вважається найтехнологічнішим рішенням на сьогоднішній момент. У даному рішенні присутній функціонал створення та пошуку заходів з подальшою їх фільтрацією. Для підтвердження відвідування система використовує зчитування по QR-коду або сканування персонального пристрою технологією NFC. Також наявна можливість відмітитись за допомогою сучасної технології сканування обличчя, що робить систему більш захищеною.

Найкраще для керування подіями, що відбуваються одночасно або за сотні миль від місця перебування організатора.

Переваги:

— велика варіативність способів відмічання гостя;

— використання радіочастотної ідентифікації;

— відслідковування статистики відвідування в реальному часі з подальшою генерацією аналізу заходу;

— створення власного дизайну значка;

— за допомогою мікросхем вбудованих до значків та наручних смуг можливо відстежувати рухи учасників під час сеансів;

— доступна оплата через значки та наручні смуги;

— управління власними сесіями.

Недоліки:

— складність створення значків, та неможливість їх передруковування;

— немає чіткої політики відносно імен користувачів;

— складне налаштування системи;

— користувацька підтримка лише електронної пошти.

2.4 Програмне рішення «Certain Arrive»

«Certain Arrive»[6] — це додаток розроблений компанією Certain і визнаний багатьма всесвітньовідомими компаніями. Особливістю виступає можливість реєстрації відвідувача, який не був заздалегідь внесений до списку. Саме рішення спрямоване більш на пришвидшення самого процесу відмічання гостей ніж на контроль над кожним з них. Це особливо корисно для подій, де основною метою є підвищення кількості відвідувачів серед яких можуть бути потенційні клієнти. Також присутній функціонал сканування QR-кодів.

Найкраще для безкоштовних подій чи заходів, на які очікується багато відвідувачів.

Переваги:

— дуже простий у користуванні спосіб реєстрації;

— доступний VIP режим оповіщення о прибутті очікуваного гостя;

— реєстрація на захід без попереднього запрошення;

— цілодобова підтримка від менеджерів;

— автентифікація через відсканований QR код.

Недоліки:

— складна навігація по додатку;

— сформований звіт незручний для використання, і потребує додаткових зусиль для зчитування;

— складний редактор шаблонів пошти користувачів;

— відсутня можливість одночасної реєстрації великої кількості людей;

— зазначення ідентифікаторів користувачів для певного заходу займає багато часу.

2.5 Програмне рішення «Zkipster»

«Zkipster»[7] — це програмне забезпечення для планування приватних заходів з обмеженою кількістю гостей. Запрошення відбуваються через систему електронної пошти з наступним підтвердженням присутності гостя. Відмічання відбувається при вході і кожен запрошений може відмічати додаткових гостей разом із собою. Система зберігає локально данні, тому раптові перебої мережі не зіпсують попередньо отриманих даних, і після відновлення з'єднання синхронізує данні. Програма гарантує надійне збереження та приватність персональних даних.

Найкраще для події, де місце проведення може мати ненадійне з'єднання.

Переваги:

— додаткові місця для кожного запрошеного;

— додаткова безпека даних гостя;

— посилання повідомлень безпосередньо до кожного гостя;

— можливість оголошувати місця гостей та відображати загальну мапу місць;

— присутня двофакторна автентифікація;

— наявна панель з усіма доступними контрольними точками та статистика переміщення гостей між ними.

Недоліки:

— нестабільна робота при поганому з'єднані з мережею.

— труднощі з освоєнням користувацького інтерфейсу.

В результаті проведеного аналізу сформовано порівняльну таблицю 2.1. на якій представлені переваги та недоліки розглянутих продуктів.

Таблиця 2.1 — Порівняння розглянутих готових рішень

Критерій	OnArrival by Cvent	Social Tables Check-in	Boomset	Certain Arrive	Zkipster
Наявність значка	+	-	+	+	+
Автентифікація через QR-код	+	+	+	+	+
Біометрична автентифікація	-	-	+	-	-
Авторизація через NFC	-	-	+	-	-
Централізоване управління	+	+	+	+	+
Можливість оплати	+	-	+	-	-
Наявний мобільний застосунок	+	+	+	+	+
Наявне Рішення для ПК	-	+	-	-	-
Можлива автономність	+	-	+	+	+
Оповіщення о прибутих гостях	+	-	+	+	+
Аналіз даних про захід	-	-	+	-	+
Доступна мапа заходу	-	+	-	-	+
Вартість за місяць, грн	2500	2500	5000	2500	Від 5000 до 10000

Підсумувавши, можна сказати, що основними умовами конкурентоспроможності майбутнього проєкту є вартість, простота використання, централізоване управління, швидкодія реагування та автономність системи. Система, яка розробляється в рамках цього проекту, буде повністю безкоштовною, матиме мінімальні кроки для проходження автентифікації, керування з одного пристрою та незалежність від необхідності доступу до мережі інтернет. Для досягнення максимальної швидкості відмічання у заході було вирішено позбавитись від всіх можливих взаємодій з зовнішніми пристроями.

Буде наявна лише підтримка графічного інтерфейсу з англійською мовою, однак планується додавання мов інших країн під час розширення. Нові функції також будуть додані пізніше. Але не обійшлось і без недоліків системи є підтримка лише однієї платформи — Android, і все ж в разі подальшої роботи над проєктом планується забезпечення підтримки платформ Windows та IOS, що буде здійснити достатньо легко завдяки обраним міжплатформеним сервісам, та загальності систем міжмережевого спілкування.

3 ФОРМУВАННЯ ВИМОГ ДО МОБІЛЬНОГО ЗАСТОСУНКУ

Після ознайомлення з інформацією про готові рішення було вирішено створити систему, функціональність якої повинна відповідати наступним вимогам:

— повинна підтримувати автентифікацію та авторизацію користувачів через пошту чи мобільний телефон;

— користувачі повинні бути незалежними одиницями, що можуть як ініціювати захід, так і бути спроможним відвідувати заходи інших користувачів;

— система повинна гарантувати контракт один телефон — одне відмічання в одному заході;

— слідувати правилу, що власник заходу не може відмітитись в ньому.

— користувач матиме змогу після відмічання в заході залишити його за власним бажанням;

— власник спроможний видаляти небажаних гостей зі свого заходу, і при цьому повинне бути належне інформування видаленого про цю дію;

— проходження основного процесу контролю відвідування повинне бути автономним, тобто без доступу до глобальної мережі Інтернет;

— система зобов’язується відображати історію заходів користувача як створених, так і відвіданих;

— наявне централізоване зберігання даних, з можливим доступом до них без підключення до глобальної мережі.

Для даної системи було обрано рішення у вигляді мобільного застосунку з хмарним сервісом. Для її побудови слід визначити критерії, яким повинні відповідати пристрій користувача та хмарний сервіс.

Вимоги до пристрою:

— операційна система Android;

— версія Android 4.1 і вище;

— необов’язкова підтримка функцій модему, таких як «точка доступу»;

— наявний Wi-Fi модуль та можливість пристрою приєднуватись до бездротових мереж.

Вимоги до сервісу:

— підтримка загальних методів автентифікації, наприклад OAuth 2.0;

— можливість перевіряти введені користувачем ідентифікатори, такі як пошта та мобільний телефон;

— надання доступу до бази даних;

— посилання сповіщень користувачам.

Згідно з описаних вимог мобільний застосунок повинен мати розділення зазначеного функціонала на окремі форми, які несуть відповідну інформацію. Це допоможе розмежувати область використання класів реалізації, що зменшить навантаження на систему, і зробить її більш продуктивною. Це полегшить користувачеві роботу з функціоналом, який його цікавить, і система стане легшою для ознайомлення.

Оскільки система вимагає авторизацію користувача, створення заходу, відмічання в ньому та перегляд історії, то рекомендується розділення інтерфейсу на наступі області.

Область автентифікації та створення користувача. Являтиме собою групу вікон з полями вводу персональних даних користувача. Вікно входу потребуватиме лише пошту користувача та його пароль, саме за цими даними буде відбуватися автентифікація користувача до мобільного застосунку. Для користувачів, що не мають ще власного облікового запису, з вікна входу буде доступна форма створення користувача, де на додачу буде вимагатися введення свого ім’я. На невідомі ще хмарному серверу пошти буде надсилатися лист підтвердження, а доти в застосунку буде відображатись вікно, яке буде блокувати доступ до основного функціонала поки пошта не підтвердиться. На вікні очікування підтвердження пошти слід надати можливість надсилання повторного листа, а також кнопку, яка буде сповіщати, що пошта підтверджена, і впускати користувача до меню.

Область меню. Являтиме собою вікно з доступними діями для користувача та мусить надати користувачеві можливість змінювати своє ім’я та функцію виходу зі свого облікового запису.

Область створення заходу. Являтиме собою вікно з внесенням інформації щодо створюваного заходу, такі як назва та опис. Кнопка початку відмічання повинна бути активна лише при наявному підключені до безпровідної локальної мережі. Всі користувачі, які відмітились в даному заході, відображатимуться на тому ж вікні.

Область з доступними заходами для відмічання. Являтиме собою вікно зі списком всіх доступних заходів в мережі, до якої підключений пристрій користувача. Після відмічання певного заходу користувач повинен мати змогу покинути його, але це лише під час активної сесії заходу.

Область історії заходів. Являтиме собою вікно з двома вкладками. Перша — для відображення створених заходів, друга — для відображення заходів відвіданих користувачем. Натомість кожне поле в списку створених заходів мусить надавати інформацію про користувачів, які відвідали його, наприклад, через список, який розгорається під вибраним заходом.

Слід звернути увагу, що при необхідності демонстрації роботи системи, використовують спеціалізовані діаграми варіантів використання або як ще називають «use-case» діаграми (кресленик ІА61.080БАК.0035 Д1), які демонструють взаємодію користувачів з системою в простому вигляді та необхідному для розуміння обсязі. Список всіх варіантів використання показує вимоги до розроблюваної інформаційної системи за допомогою яких можна створити технічне завдання, тому така діаграма є важливим засобом для аналізу вимог до системи.

4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

Під час вибору технологій слід звертати увагу на тенденції їх розвитку, оскільки це дасть чітко зрозуміти наскільки прогресивним буде рішення через пару років. Актуальність обраних технологій зробить програмне забезпечення конкурентоспроможним на поточному ринку. Та найголовнішим критерієм є відповідність технологій розробки до поставлених вимог.

4.1 Клієнт-серверна архітектура

Клієнт-серверна архітектура[8] розв’язує задачі централізованого надання послуг та розподіляє мережеве навантаження, її загальна схема наведена на рисунку 4.1. Клієнти виступають споживачами цих послуг. Це рішення доцільно використовувати при системах з великою кількістю користувачів, які одночасно можуть звертатися за даними до сервера, а саме цього треба досягти, оскільки заходи мобільного застосунку налічуватимуть сотні відвідувачів.

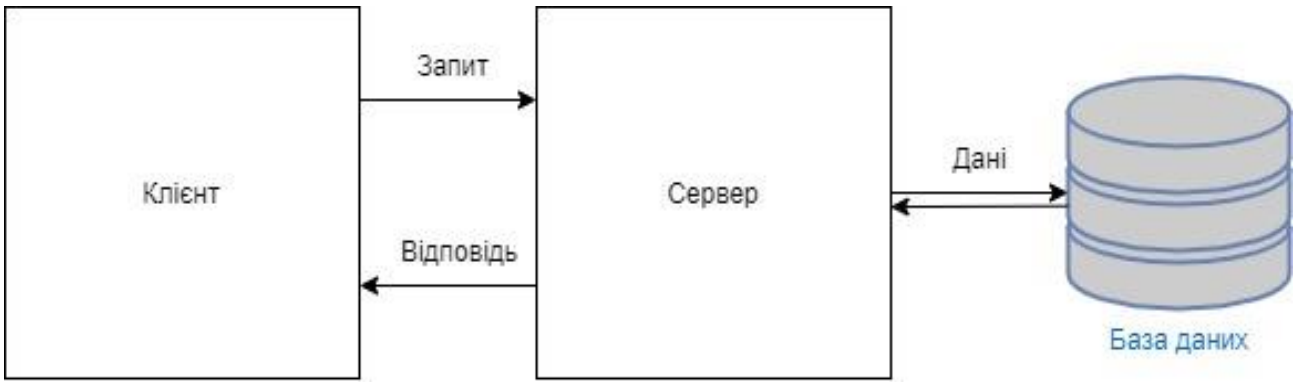

Рисунок 4.1 — Загальна діаграма клієнт-серверної архітектури

Сервер — це програмний компонент системи, яка забезпечує доступ до загальних ресурсів та керує ними.

Клієнт — це програмний компонент системи, що виступає отримувачем ресурсів та сервісів, які дає сервер.

Основні задачі серверу:

— обробка даних своєю обчислювальною потужністю;

— захищений доступ до даних та ресурсів;

— управління трафіком.

Розробка буде вестись під операційну систему Android, яка слугуватиме як сервером, так і клієнтом. Тому для більшої стабільності системи організовується розподіл на серверну та клієнтську операційні системи. Це слугує для розподілення повноважень між сервером та клієнтом мережі. Зокрема серверна операційна система більшою мірою базується на управлінні ресурсами, натомість клієнтська - на виконання різних процесів з максимальною швидкістю та ефективністю.

Серверні операційні системи можуть виконувати як свої прямі обов’язки так і виступати в ролі клієнта. Так пристрій сервер виступатиме клієнтом для хмарного сервісу Firebase. Для виконання прямих обов’язків серверна операційна система повинна видавати високі показники швидкодії, і це реалізується за допомогою багатопоточності, багатозадачності та багатопроцесорності.

Багатопоточність[9] виділяють як основну і найнеобхіднішу особливість серверних операційних систем. Саме завдяки цьому досягається повне використання процесорного часу, що супроводжується максимальною ефективністю системи.

При наявній функції багатозадачності[10] у серверній операційній системі, час процесора розподіляється між кожним процесом, хоча з точки зору користувача виглядає наче процеси виконуються паралельно. Це реалізовується завдяки високій швидкості виконання обчислювальних процесів, а також властивістю сортувати інтервали часу процесів. Даний принцип значно покращує продуктивність роботи серверу і системи в цілому. Ефективність даного підходу ще більше зростає при реалізації даного принципу на пристрої клієнта, що дозволить зробити клієнт-серверну взаємодію більш динамічною і продуктивною,

та дозволяє усунути випадки, коли пристрій клієнта перебуває в стані очікування завершення обробки даних сервером, і не спроможний виконувати інших функцій.

Багатопроцесорна операційна система базується на масштабуванні кількості процесорів, тим самим покращуючи продуктивність, завдяки розподіленню задач між декількома багатопоточними і базатозадачними процесорами. Але цей підхід не розповсюджується на операційні системи Android.

Роблячи підсумок, можна сказати, що клієнт-серверна архітектура повністю покриває потреби мобільного застосунку. Вона забезпечить централізоване зберігання даних, що усуне можливість втрати персональної інформації. І також зменшить обсяг самого мобільного застосунку на персональному пристрої.

4.2 Хмарний сервіс Firebase

Для реалізації клієнт-серверної архітектури де мобільний застосунок буде виступати клієнтом потрібен сервер. Було обрано рішення від Firebase.

Firebase[11] — це BaaS(Backend-as-a-Service) що функціонує як хмарний сервер. І він виступає як набір інструментів для вдосконалення та розробки мобільних застосунків, інтернет-сайтів та комп'ютерних програм. І надає значну частину найпотрібніших послуг. Це зроблено для поліпшення роботи розробників, і дозволяє їм зосередитися на вдосконаленні свого програмного рішення. До сервісу входять такі послуги, як аналітика, автентифікація, бази даних, конфігурація, зберігання файлів, передача повідомлень. Сервер розміщуються у хмарі та може масштабуватися без особливих зусиль з боку розробників.

4.2.1 Firebase автентифікація

Автентифікація Firebase має вбудовану систему реєстрації через пароль електронну пошту чи телефон. Віна також підтримує OAuth 2.0[12] для Google, Facebook, Twitter та GitHub. Розробник також можете інтегрувати власні системи автентифікації з Firebase Authentication, щоб надати користувачам доступ до даних, не змушуючи їх створювати обліковий запис за межами наявних систем.

Firebase Authentication надає персональну консоль для розробників, де показується статистика по кількості користувачів, та їх активності.

OAuth - це відкритий стандартний протокол авторизації, який надає програмам можливість здійснювати автентифікацію через довірені програми. OAuth не ділиться даними паролів, а використовує маркери авторизації для підтвердження ідентичності між споживачами та постачальниками послуг. Основна задача якої схвалювати одну програму, яка взаємодіє з іншою від імені користувача, не надаючи пароль. На рисунку 4.2 представлена загальна схема протоколу.

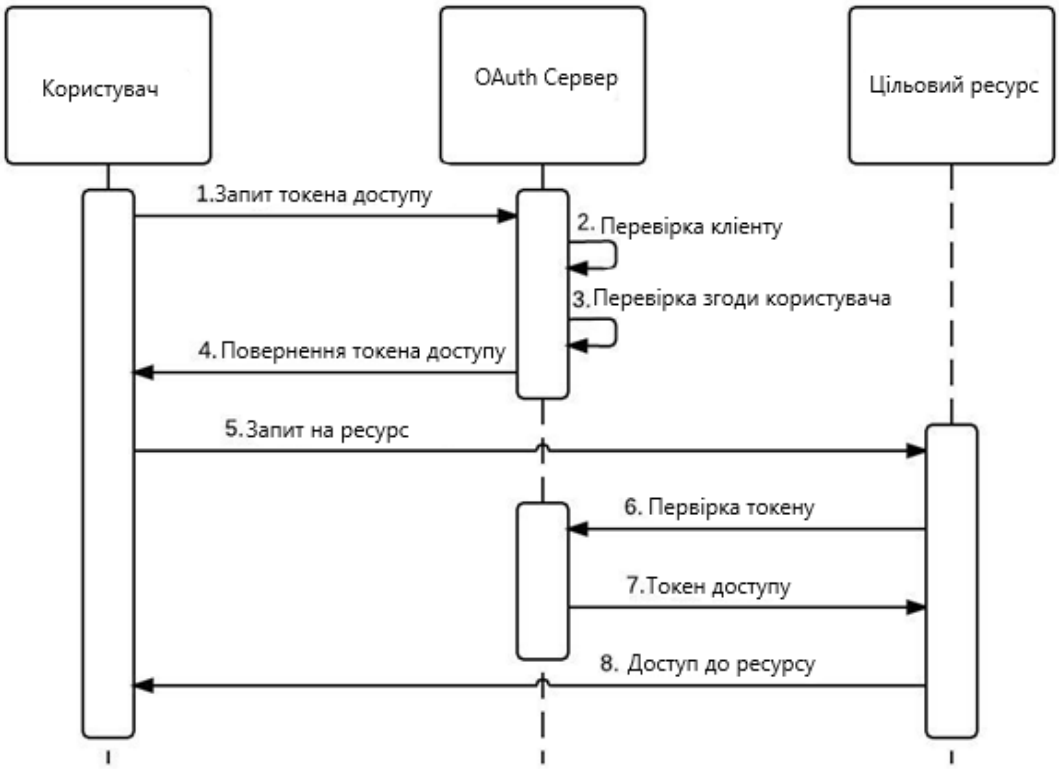

Рисунок 4.2 — Діаграма доступу до ресурсу через OAuth

Firebase також дозволяє створювати анонімні сеанси автентифікації, які зазвичай використовуються для збереження невеликої кількості даних під час очікування клієнта для автентифікації за допомогою методу постійної автентифікації. Ці анонімні сеанси можна налаштувати на тривалі період в декілька днів, тижнів, місяців, навіть років, поки користувач не ввійде в систему за допомогою методу постійного входу або очищення кеш-пам’яті браузера.

Мобільні застосунки часто використовують локальні сховища даних, для виконання подібних завдань. Наприклад, додаток для кошика може створити анонімний сеанс автентифікації для кожного користувача, який щось додає у свій кошик. Додаток кошик попросить користувача створити обліковий запис користувача для оформлення замовлення, і в цей момент кошик буде зберігатися в обліковому записі нового користувача, а анонімний сеанс буде знищений.

Переваги автентифікації через Firebase:

— Firebase дозволяє швидко розпочати роботу з API авторизації;

— контролювання доступу за допомогою рольової моделі;

— покладається на перевірку персональних даних користувачів;

— можливість створення API REST через Firebase Functions та додавання власних обробників автентифікаційних запитів.

4.2.2 База даних Firebase

База даних реального часу Firebase дозволяє безпечно зберігати дані на хмарних серверах Google і синхронізувати їх у реальному часі для всіх клієнтів, що під’єднані до однієї бази даних.

Основна мета системи — забезпечити безпечний, надійний і швидкий спосіб синхронізації даних. Система баз даних також призначена для масштабування, що дасть можливість підтримувати мільйони користувачів.

База даних називається реального часу, оскільки швидкість, з якою синхронізуються дані між клієнтами, майже близька до реального часу, і обмежується фізичним обмеженням передачі даних через Інтернет. Зміна даних про одного клієнта поширюється на іншого, візуально непомітно.

Система також забезпечує збереження даних локально, тим самим даючи змогу зберегти доступ до даних, навіть коли пристрій перебуває в автономному режимі. Коли з’єднання буде відновлено, локальні дані автоматично синхронізуються та об'єднуються з віддаленими даними.

Firebase використовує базу даних типу NoSQL. Як випливає з назви, це означає, що дані не зберігаються в таблицях і рядках, що знаходяться в системах

управління реляційними базами даних, такими як Oracle Database або Microsoft SQL Server. Також дані не отримуються за допомогою запитів Structured Query Language (SQL). Натомість дані зберігаються у вигляді об’єкта JSON, який є легким як для людей, так і для програмного забезпечення для читання, запису та розуміння.

JSON (JavaScript Object Notation) — це простий формат обміну даними між, який представлений у текстовому вигляді і виступає відкритим стандартом файлу. Формат читабельний для людини, а також виступає офіційним форматом обміну в мережі Інтернет. Він був запозичений від мови програмування JavaScript, але зараз він є універсальним та може використовуватися будь-якою мовою програмування. Структура формату JSON має вигляд дерева.

JSON може виступати однією з двох типів структур:

— набір пар ключ-значення;

— впорядкований набір значень.

Підсумувавши всю інформацію про базу даних реального часу Firebase можна зробити висновок, що вона достатньо оптимізована для роботи з мобільними застосунками, дуже легка і досить швидка. Написання коду для роботи з нею зводитеся до мінімуму, що стало причиною вибору її для даного проєкту.

4.2.3 Сервіс сповіщення Firebase

Push-сповіщення[13] — це головний засіб, для повідомлення користувача про важливу інформацію. Являє собою невеликі повідомлення, які направлені досягти цільової аудиторії в будь-який час. Вони виступають частиною моделі «публікація-підписка» і беруть участь у клієнт-серверному спілкуванні.

Платформа хмарних повідомлень Firebase — це безкоштовна мобільна служба сповіщень від Google, яка дозволяє розробникам застосувань надсилати сповіщення з Google Cloud Messaging серверів своїм користувачам. Ця процедура інформування користувача відбувається в режимі реального часу. Вона працює за принципом потокової передачі повідомлень з серверів FCM на додаток користувача та висхідних повідомлень із програм користувача назад.

Для надсилання та отримання повідомлень за допомогою Firebase потрібні два елементи: відповідний сервер, на якому можна будувати, направляти та надсилати повідомлення та додаток для Android, iOS або веб-клієнтів для отримання повідомлень. За допомогою Firebase розробники можуть надсилати користувачам два типи повідомлень: повідомлення даних та повідомлення сповіщень.

Повідомлення про сповіщення відображаються на пристрої користувача Firebase від імені програми, тоді як повідомлення даних безпосередньо обробляється додатком, який відповідає за доставлення повідомлення користувачеві.

Firebase здатна доставляти повідомлення програмам через націлювання на повідомлення трьома способами: на окремі пристрої, на пристрої, які підписані на теми та на групи пристроїв. Розробники можуть створювати повідомлення у композиторі повідомлень, які можуть надсилати цільові повідомлення певним сегментам користувачів. Ці повідомлення повністю інтегровані з Firebase Analytics, яка відстежує залучення користувачів.

Цей хмарний сервіс повністю підтримується та інтенсивно просувається компанією Google. Він повність покриває мої потреби що до розробки мобільного застосунку.

4.3 Мова програмування Java

Java[14] — це строго типізована об’єктно-орієнтована мова програмування розроблена компанією Sun Microsystems, згодом придбана компанією Oracle, що і займається підтримкою зараз. Весь програмний код компілюється в bytecode[15] і працює під JVM. Використовується для великих підприємницьких рішень.

Особливості мови програмування:

— компільований код Java може працювати на всіх платформах, які підтримують її, не потребуючи рекомпіляції;

— має досить легкий синтаксис, і легко освоюється початківцями;

— вважається безпечною через роботу програми всередині JVM, де віруси не мають доступу до системи;

— стабільність, вся робота з пам’яттю забезпечуються внутрішніми засобами;

— швидкодійна внаслідок близькості bytecode до машинного;

— ефективна багатопоточність внаслідок економії пам’яті для кожного нового потоку;

— легка підтримка інших програмних мов;

— Java є нейтральною для архітектури, оскільки немає функцій, залежних від реалізації;

— абстрактність підходу програмування.

Оскільки Android оснований на операційному ядрі Linux та особливій реалізації JVM[16]. Java хоч і не вважається сучасною мовою програмування, однак саме вона забезпечить покриття більшої кількості користувачів шляхом підтримки ще ранніх версій Android. Завдяки її стабільності і захищеності застосунок буде позбавлений від великої кількості системних помилок, а дані користувача з меншою ймовірністю вкрадуть вірусні програми. Java ідеально підходить для розробки мобільного застосунку, завдяки архітектурній незалежності, швидкодії та роботі з пам’яттю.

4.4 Програмний інтерфейс Socket

Програмні інтерфейси[17] стали невіддільною частиною будь-якого програмного рішення. Їх ще називають Application programming interface(API), вони значно облегшують розробку програми, яка потребує зв'язок частин рішення, розташованих на різних пристроях, з’єднаних мережевим сполученням. Клієнт-серверна архітектура використовує саме це рішення.

API — це сукупність функцій та процедур, що дозволяють створювати додатки, які отримують доступ до даних та особливостей інших додатків, служб або операційних системи.

Кожному пристрою, який підключений до комп'ютерної мережі, присвоюється IP-адреса Інтернет-протоколу, який є динамічним, і присвоюється лише на певний час з'єднання. До кожної програми, що працює на такому пристрої, можна отримати доступ через аналогічну пару номерів, яка називається IP-адресою та номером порту. Ця пара відома як програмний інтерфейс Socket.

Всього доступно чотири типи сокетів: Stream Sockets, Datagram Sockets, Raw Sockets, Sequenced Packet Sockets. Перші два є найпопулярнішими і найчастіше використовуються, а останні два — вспоміжні і використовуються рідко. Передбачається, що процеси спілкуються лише між сокетами одного типу, але немає обмеження, яке б перешкоджало зв'язку між сокетами різних типів.

Stream Sockets гарантують доставлення в мережевому середовищі. Вони використовують TCP для передачі даних. Якщо доставлення неможлива, відправник отримує індикатор помилки. Записи даних не мають меж.

Datagram Sockets не гарантують доставлення у мережевому середовищі. Вони не підтримують з'єднання, як у Stream Sockets, створюється пакет з інформацією про адресу призначення та відправляється. Даний тип використовує UDP пакети.

Raw Sockets надають користувачам доступ до базових протоколів зв'язку, які підтримують абстракцію сокета. Вони зазвичай орієнтовані на Datagram Sockets, хоча їх точні характеристики залежать від інтерфейсу, наданого протоколом. Ці сокети не призначені для загального користувача, вони були надані в основному тим, хто зацікавлений у розробці нових протоколів зв'язку або для отримання доступу до деяких певних протоколів для криптовалют.

Sequenced Packet Sockets схожі на Stream Sockets, за винятком того, що межі запису збережені. Цей інтерфейс надається лише як частина абстракції Network Systems і є дуже важливим у більшості серйозних його застосувань. Даний тип

сокету дозволяє користувачеві маніпулювати заголовками пакетів або протоколом Internet Datagram Protocol (IDP).

В цілому API часто описується як будь-який тип інтерфейсу загального підключення до програми. Однак останнім часом сучасний API набув деяких характеристик, які роблять їх надзвичайно цінними та корисними:

а) API дотримуються стандартів, які є зручними для розробників, легко доступними та зрозумілими широкій аудиторії.

б) До них ставляться більше як до продуктів, ніж до коду. Вони розроблені для споживання для конкретних аудиторій, наприклад розробниками мобільних застосунків, також документально підтверджені та розроблені так, щоб користувачі могли мати певні очікування щодо його обслуговування.

в) Оскільки вони більш стандартизовані, мають набагато надійнішу політику щодо безпеки та управління, а також контролюють їх ефективність та масштаб.

г) Як і будь-яка інша частина розробленого програмного забезпечення, сучасний API має власний життєвий цикл розробки програмного забезпечення, проєктування, тестування, побудови, управління версіями.

4.5 Бібліотека сканування штрих-кодів

Штрих-код[18] — це формат запису даних, який зручний для зчитування спеціальним пристроєм. Найбільш розповсюджена форма штрих-коду представлена у вигляді послідовності смуг різної товщини.

Під потреби зчитування набору даних фотокамерою мобільного телефона була обрана технологія QR-код, що є двовимірного версією штрих-коду. Ця технологія відрізняється швидшим часом відгуку, та можливість вміщувати більші обсяги інформації.

Для данного мобільного застосунку була обрана бібліотека ZXing. Вона надає можливість створювати QR-коди у вигляді звичайних картинок відповідно до вказаного тексту. Наявне регулювання розміру картинки. Також спроможна зчитувати як звичайні штрих-коди так і QR-коди. Зчитування відбувається за

допомогою камери телефона, що створюється в окремому вікні. Бібліотека має надає можливість широко налаштовувати режим сканування, задаючи розміри штрих-коду, його положення та тип. Вона дуже легко інтегрується до проектів під Android, і потребує мінімального написання коду.

Ця бібліотека достатньо покриває потреби роботи системи, з усіма необхідними функціями. Також дозволяє використовувати сучасний формат запису даних.

4.6 Висновки до розділу

Під потреби централізованої обробки даних мобільного застосунку була обрана клієнт-серверна архітектура на базі хмарного сервісу Firebase. Розробка буде вестись мовою програмування Java для платформи Android. Під управління базою даних обраний сервіс Firebase Realtime Database, що має легку структуру зберігання даних та надає можливість безперервного оновлення інформації. База даних формату NoSQL. Текстовий формат обміну даними JSON слугуватиме для обміну між пристроями, і відсилання на сервер. Підтримка протоколу OAuth 2.0 для автентифікації користувачів. Інтеграція бібліотеки ZXing для створення та зчитування QR-кодів. Для більш швидкого спілкування між пристроями обрана технологія програмного інтерфейсу — Socket, та сервіс сповіщення Firebase Cloud Messaging.

5 РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

5.1 Структура проєкту

В структурі системи можна виділити наступні основні частини: користувацький інтерфейс, автентифікація, база даних, сокети передачі даних та сервіс сповіщення. На стороні користувацького інтерфейсу реалізовується найбільш видима для користувача функціональність, з якою йому буде надана можливість працювати. В такому випадку це мобільне застосування. Серверна частина реалізує необхідну автентифікацію, сповіщення і також виконує синхронізацію бази даних відповідно користувацьких змін. Всі вони розділені по цільовим каталогам, які представлені на рисунку 5.1.

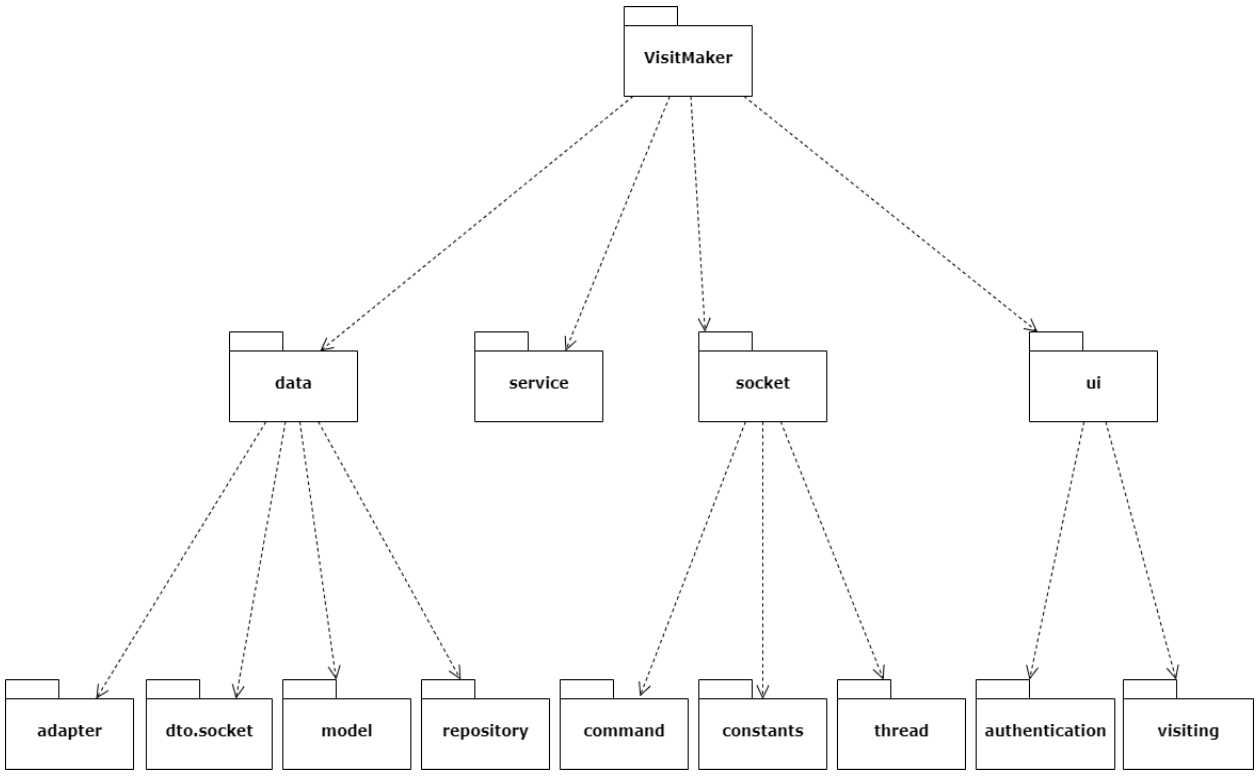

Рисунок 5.1 — Структура проєктуємого мобільного застосунку

Каталог data містить основні класи роботи та представлення даних. Всі операції модифікацій, відправки, збереження виконуються саме над цими класами. Нижче описані задачі покладені на кожний підкаталог детальніше:

а) Класи підкаталогу adapter призначені для корекції необхідних моделей відносно потреб користувацького інтерфейсу. Таким чином облегшується для користувача робота з моделями, і натомість такий підхід пришвидшує роботу графічного відображення на екрані. Їх представлення можна знайти на діаграмі класів ІА61.080БАК.005 Д1.

б) Класи підкаталогу dto.socket призначені для роботи з даними, які необхідно відправити через мережу. Сама назва Data Transfer Object вказує на це. Вони слугують своєрідною обгорткою над моделями, і містять в собі додаткову інформацію, корисну для сторони, що приймає.

в) Класи підкаталогу model призначені для представлення моделей, саме вони слугують сутностями для бази даних, і фігурують головними елементами для класів з підкаталогу adapter з пункту а.

г) Класи підкаталогу repository призначені для роботи з сервером, і виконують функції створення, отримання, редагування та видалення відповідних сутностей. І кожен клас відповідає лише за свою модель.

Каталог service містить в собі класи, які надають допоміжні методи для роботи системи. Це в основному класи роботи з інтерфейсами глобальної мережі, посилання сповіщень та підтримка роботи з поточним користувачем.

Каталог socket містить в собі класи роботи з програмним інтерфейсом Socket, і призначені для виконання в асинхронному режимі окремо від інтерфейсу користувача. Нижче описані задачі покладені на кожний підкаталог детальніше:

а) Класи підкаталогу thread являються головними під час передачі даних через локальну мережу. Всі вони являються окремими потоками, і виконуються паралельно від користувацького інтерфейсу. Основна їх ціль це створення окремих команд, та відстеження відповідей на них.

б) Класи підкаталогу constants призначені для надання статичних констант для роботи сокетів.

в) Класи підкаталогу command слугують для посилання через локальну мережу. Вони реалізують поведінковий шаблон проєктування «Command», і це

дозволяє на приймальній стороні не визначати операцію по статичному тексту, а безпосередньо виконувати отриману команду. Саме ці команди передаються класами з пункту а.

Каталог ui містить собі класи, які слугують для контролю користувацьким інтерфейсом. І саме вони оброблюють операції користувача. Для зручності роботи, вони були розділені на два підкаталоги: authentication та visiting, які відповідають за автентифікацію та організацію відмічання відповідно.

5.2 Автентифікація користувача

Автентифікація користувача потребує внесення користувачем користувацького ідентифікатора. Для даної системи цим ідентифікатором виступає пошта користувача. Це виконується для підтвердження надійності користувача, і слугує підтвердженням унікальності облікового запису.

Створення користувача здійснюється після внесення наступних даних: електронної адреси, ім’я та паролю. Але доступ до мобільного додатку надається лише після підтвердження електронної пошти, а це потребує наявного доступу до мережі інтернет. Таке рішення буде гарантувати, що пошта дійсно належить людині, яка проходить реєстрацію. Натомість система буде захищена від DoS атак через перенавантаження надмірною кількістю користувачів. Подальший вхід в систему потребуватиме лише пошти та пароля.

Сама автентифікація відбувається через сервіси Google, а саме Firebase Authentication. Саме він організовує відправлення листа на електронну пошту і відповідає за збереження статусу користувача в системі. Цей сервіс самостійно виконує перевірку введених персональних даних користувача і формує для нього ідентифікатор. Схема цього процесу представлена на рисунку 5.2.

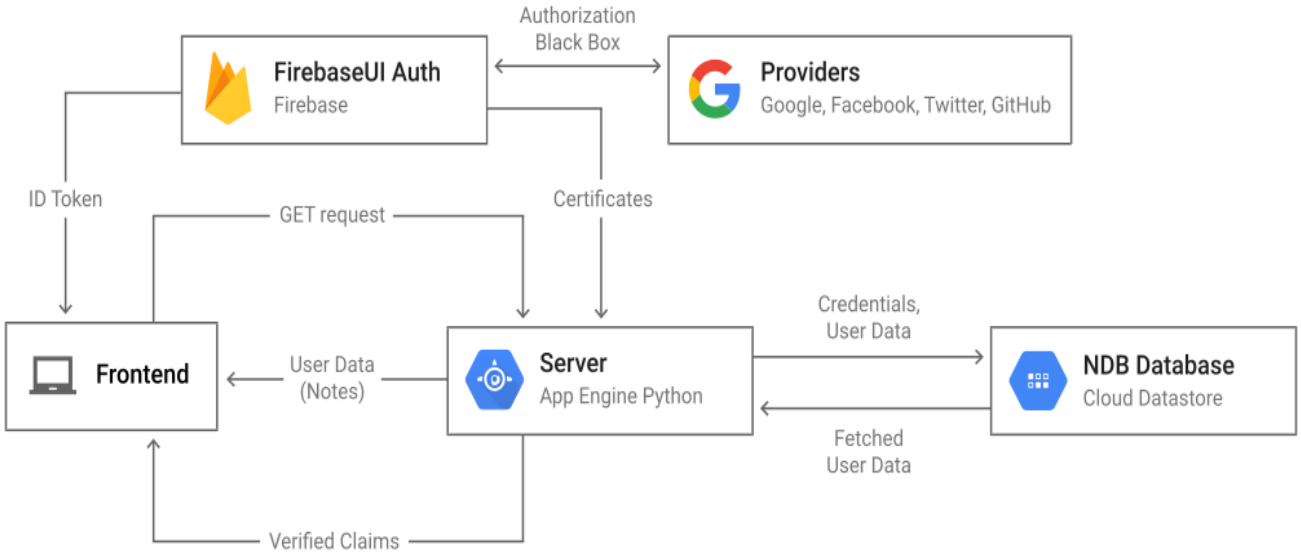

Рисунок 5.2 — Схема автентифікації через Firebase[19]

За роботу з користувачем відповідає UserManager клас, програмний код якого представлений у додатку А, і що надає наступні методи:

а) login() — метод входу до наявного облікового запису.

б) signIn() — метод створення нового облікового запису.

в) getCurrentUser() — метод отримання поточного користувача.

г) isUserEnailVerified() — метод перевірки, чи підтвердив користувач пошту.

ґ) userIsAuthenticated() — метод перевірки, чи наявний поточний користувач.

д) reloadUser() — метод оновлення даних про користувача.

е) signOut() — метод виходу з облікового запису.

Використання одного користувача на різних пристроях також підтримується, при кожному підключенні до Інтернету дані синхронізуються хмарним сховищем.

5.3 Модель бази даних

Для розв’язання поставленої задачі до проєкту системі необхідно лише дві моделі об’єктів для зберігання в базі даних. Це сутність користувача та сутність заходу. Оскільки дані про користувача вже зберігаються сервісом Firebase, то в базі будуть зберігатися лише ідентифікатор користувача та надлишкові дані. Ці

дві сутності перебувають відразу у двох відношеннях «один до багатьох» та «багато до багатьох», які представлені на рисунку 5.3.

Відношення «багато до багатьох» призначене для задач розподілу відвідувачів та заходів. Так будь-який користувач може відмітитись в безлічі заходів, і відповідно кожен захід здатний в собі містити велику кількість відвідувачів. Це відношення потребує додаткової сполучної ланки між двома цими сутностями.

Відношення «один до багатьох» призначене для задач розподілу власника та створеного ним заходу. Таким чином зв’язок між одним користувачем та всіма його заходами встановлюється через ідентифікатор користувача, який присутній у кожній сутності заходу.

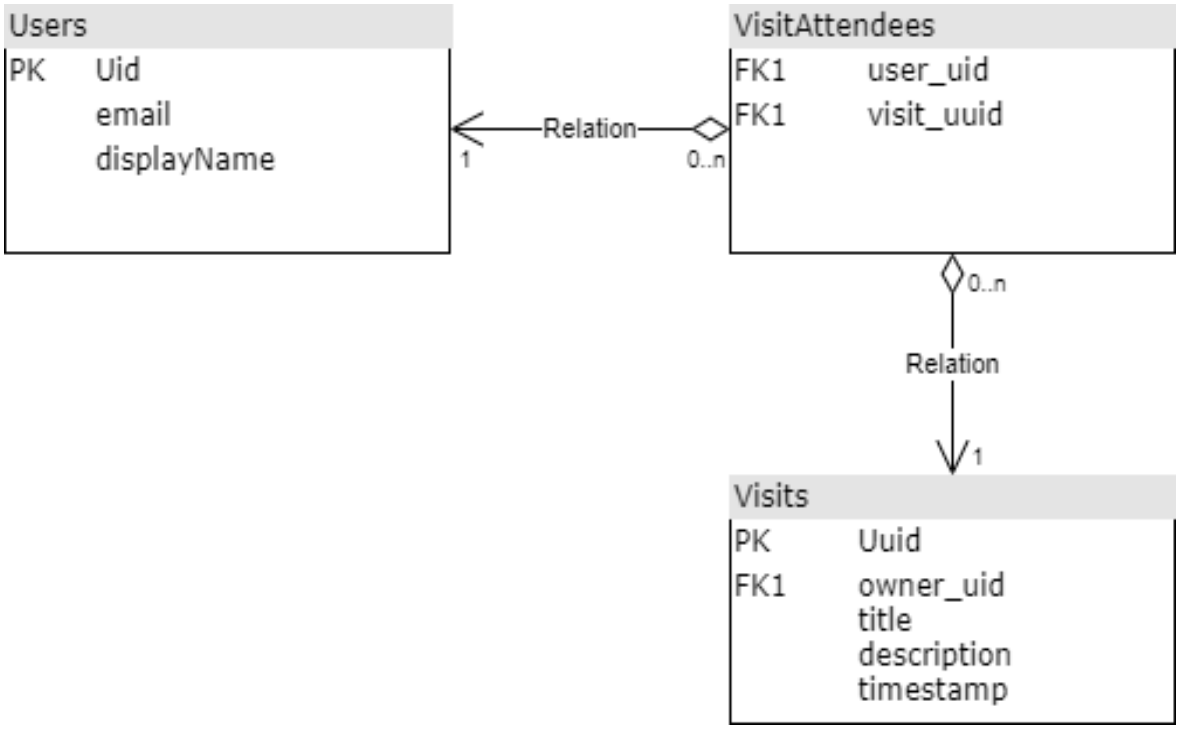

Рисунок 5.3 — схема відношення сутностей бази даних

При описі технологій розробки було вказано, що системою використовується Firebase Realtime Database. Це типова NoSQL база даних, яка використовує за основу JSON, і також має автоматичну синхронізацію при повторних підключеннях до глобальної мережі. Так як і при автентифікації, дані зберігаються за допомогою сервісу Firebase.

Приклад структури бази даних відповідно до схеми на рисунку 5.3 представлений на рисунку 5.4.

```
{
  "users" : {
    uid : {
      "displayName" : "Maksym",
      "email" : "example@mail.com"
    }
  },
  "visit_attendees" : {
    uuid : {
      uid : "Visitor1"
      uid : "Visitor2"
      uid : "Visitor3"
    }
  },
  "visits" : {
    uuid : {
      "description" : "Decription",
      "ownerId" : uid,
      "timestamp" : 1589032911747,
      "title" : "Visit"
    }
  }
}
```

Рисунок 5.4 — Приклад зберігання даних

Кожна сутність представлена у вигляді пари ключ-значення. Ключ повинен бути унікальним для всіх записів. Таким чином отримується три кореневих об’єкти, які є аналогами таблиць в реляційних базах даних.

Кожний екземпляр сутності користувача зображаться парою: ключ — унікальний ідентифікатор, який генерується на етапі створення облікового запису сервісом Firebase Authentication, і значення — додаткові поля користувача, такі як ім’я і пошта.

Сутність «visits» зображаться парою: ключ — UUID, що генерується кожний раз, коли користувач створює захід, і значення — інформація по заходу, а саме назва, опис, міткою часу та унікальним ідентифікатором користувача.

Сутність «visit_attendees» використовується для реалізації відношення «багато до багатьох» і зображаться парою: ключ — унікальний глобальний ідентифікатор створеного заходу, і значення — сутності користувачів, які відвідали цей захід у вигляді: ідентифікатор користувача та ім’я відвідувача.

Всі розроблені сутності реалізують інтерфейс «Serializable», що дозволяє їх конвертувати у зручний для передачі формат, оскільки вони будуть брати участь у подальшій їх передачі до хмарного сервісу. Описані сутності можна побачити на діаграмі класів ІА61.080БАК.005 Д1.

5.4 Передача даних

Система контролю відвідування функціонує через локальну безпровідну мережу Wi-Fi стандарту 802.11. Альтернативним рішенням слугує створення власної точки доступу, що гарантує швидшу маршрутизацію і може бути використаним в місцях і з відсутнім безпровідним маршрутизатором. Обидва варіанти підтримують одночасне підключення 250 користувачів, і від одного до чотирьох з’єднань зарезервовані під власне використання. Створена безпровідна мережа не зобов’язана бути підключена до глобальної мережі Internet.

Передача даних відбувається на транспортному рівні мережевої моделі TCP/IP. Для кожного учасника заходу формується персональна кінцева точка — Socket, яка являє собою IP адресу, яка присвоюється бездротовим маршрутизатором для кожного пристрою, та TCP порт. Так для обміну даними формується чотири кінцеві точки.

Перша кінцева точка — DatagramSocket пристрою сервера для порту 8888, що встановлюється режим трансляції. Вона відправляє дані про створений захід всім учасникам локальної мережі. Ці дані передаються в форматі UDP, вони достатньо невеликі, щоб не забивати локальну мережу, але не гарантують обов’язкове доставлення пакету. Пакет також в собі містить дані про точку доступу, через яку буде здійсненне основне спілкування. Код даного класу знаходиться в додатку Б.

Друга кінцева точка — DatagramSocket пристрою клієнта для порту 8888. Він призначений для прослуховування мережі на UDP пакети, і в разі отримання даних від пристрою сервера, показує користувачеві доступний захід для відвідування. Код даного класу знаходиться в додатку В.

Третя кінцева точка — ClientSocket пристрою клієнта для порту 8080, що формує TCP пакети з інформацією про поточного користувача. Також прикріпляються додаткові дані про унікальний ідентифікатор пристрою та персональний ідентифікатор користувача. На відміну від UDP, TCP гарантує доставлення пакетів до кінцевої точки, що запобігає посиланню зайвих пакетів даних. Код даного класу знаходиться в додатку Г.

Четверта кінцева точка — ServerSocket пристрою сервера, що працює на порті 8080. Саме адреса цієї кінцевої точки розсилається першим DatagramSocket. Слугує для приймання TCP пакетів від ClientSocket з інформацією о користувачах, які відвідали захід. Використовує ідентифікатор пристрою та персональний ідентифікатор користувача для запобігання відмічання декількох користувачів та власника в одному створеному заході. Код даного класу знаходиться в додатку Д.

Вище описані сокети реалізовані у вигляді п’яти класів. Три з яких виступають незалежними потоками, а інші два — кінцевими процесами. Окремі потоки слугують для неперервного обміну даними між пристроями під час проведення заходу. Вони створюються відповідними вікнами програми, і функціонують на паралельно до користувацького інтерфейсу. Саме вони ініціюють транслювання і отримання UDP/TCP пакетів. Своєю чергою кінцеві процеси слугують для посилання одиничного пакету, і вони сповіщають користувача про успішність його дії, прикладом слугують: відмічання в заході та видалення гостя зі списку. Розроблені класи можна побачити на діаграмі класів ІА61.080БАК.005 Д1.

Разом ці кінцеві точки будують повноцінну систему спілкування між двома пристроями, і привносять до мобільного застосунку всі їх переваги: швидкість передачі, економність, та надійність.

5.5 Сервіс сповіщення

Функціонал видалення користувачів, які відвідали захід, забезпечує оповіщення видаленого про цю дію. Це супроводжується звуковим сигналом та текстовим повідомленням, яке містить інформацію про захід з якого було видалено гостя. Нотифікація здійснюється двома способами: через локальну мережу, та через глобальну мережу Internet.

Через локальну мережу пристрій-сервер намагається з'єднатися з адресою, яка прийшла разом з даними про користувача. Це на випадок, коли ще користувач не залишив мережу, і доцільніше відправити йому повідомлення саме через неї. Пошук доступності користувача відбувається протягом трьох секунд.

Через мережу Інтернет здійснюються в разі невдачі пошуку користувача в локальній мережі. У цьому випадку на хмарний сервер Firebase формується HTTP запит, який містить в собі персональний ідентифікатор користувача, і відповідно на нього надсилається повідомлення.

Весь механізм обробки надісланих сповіщень реалізований в класі VisitMakerMessagingService, який показаний на рисунку 5.1. Він виступає окремим процесом в операційній системі, і оброблює повідомлення до мобільного застосунку, і користувач отримує повідомлення від імені цього додатку.

5.6 Користувацький інтерфейс

Відкриваючи мобільний застосунок користувач потрапляє на вікно, яке залежить від статусу автентифікації. Доступні варіанти вікон відповідно до цього статусу представлені нижче.

Для неавтентифікованого користувача:

— вхід в систему;

— реєстрація в системі;

— сторінка перевірки пошти.

Для автентифікованого користувача:

— створити захід;

— відмітитись в заході;

— переглянути історію;

— редагувати ім’я.

Кожне вікно виступає окремим XML файлом, яке контролюється відповідним Java класом. Далі розглянутий детальний опис кожного вікна з описом присутніх елементів.

Сторінка входу налічує в собі два текстових поля, одне для пошти користувача, інше для пароля користувача. Для підтвердження введених даних потрібно натиснути кнопку «Log In» або кнопку Enter на клавіатурі. Після чого користувач потрапить до меню мобільного застосунку, або ж на вікно підтвердження пошти, якщо він раніше цього не зробив. Також наявна кнопка переходу на вікно реєстрації.

Сторінка реєстрації налічує в собі три текстових поля, одне для пошти користувача, друге для ім’я користувача, останнє для пароля користувача. Для виконання реєстрації необхідно натиснути кнопку «Sign In» або кнопку Enter на клавіатурі. Після чого користувач потрапить до меню мобільного застосунку, або ж на вікно підтвердження пошти, якщо він раніше цього не зробив.

Вікно підтвердження пошти призначене для блокування входу до системи, доки користувач ще не підтвердив пошту. По середині цього вікна розташоване повідомлення куди був направлений лист. Також на ньому присутні дві кнопки, перша - для входу до головного меню, друга - для повторного надсилання листа на пошту.

Вікно головного меню призначене для розділення функціональних частин програми, і містить в собі кнопки переходу до наступних користувацьких вікон: створення заходу, відмічання в заході, перегляд історії. Додатковою функціональністю виступає меню налаштувань, через яке користувач може змінити ім’я облікового запису або вийти з нього.

Вікно створення заходу, яке представлене на рисунку 5.5, налічує в собі два текстових поля: перше для назви заходу, яке є обов’язковим, друге для опису до заходу. Кнопка початку відмічання буде активна лише у випадку підключеного телефона до локальної мережі, і після початку відмічання таж кнопка завершить даний процес.

Рисунок 5.5 — Вікно створення заходу

Вікно відмічання в заході налічує в собі лише список з доступними заходами в локальній мережі. Як видно з рисунку 5.6, кожний захід містить інформацію про власника, назву самого заходу і опис. І біля кожного заходу наявна кнопка для відмічання в ньому.

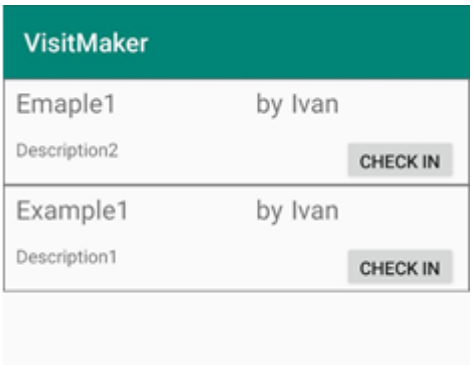

Рисунок 5.6 — Вікно відмічання в заході

Вікно перегляду історії, яке представлене на рисунку 5.7, розділене на дві вкладки: перша для зображення створених заходів, друга для зображення історії відвідування. Для кожного заходу з першої вкладки доступний виїжджаючий список з усіма його гостями.

Рисунок 5.7 — Вікно перегляду історії

5.7 Топологія системи

Для роботи автоматизованої системи контролю відвідування заходів потрібне формування відповідної схеми мережі. Відповідно до цього були розроблені можливі топології мережі. Для системи задовільні два стани мережі: з окремим мережевим маршрутизатором і з пристроєм сервером, що виступає мережевим маршрутизатором.

Топологія з окремим мережевим маршрутизатором потребує наявного на місці проведення заходу окремого Wi-Fi маршрутизатора. В такому випадку всі охочі відмітитись у заході муситимуть під'єднатися до цього маршрутизатора. Це дозволяє системі покривати більші площі заходу, з більшою кількістю людей. Однак потребує наявності додаткового обладнання у вигляді Wi-Fi маршрутизатора. На рисунку 5.8 представлена топологія даної мережі.

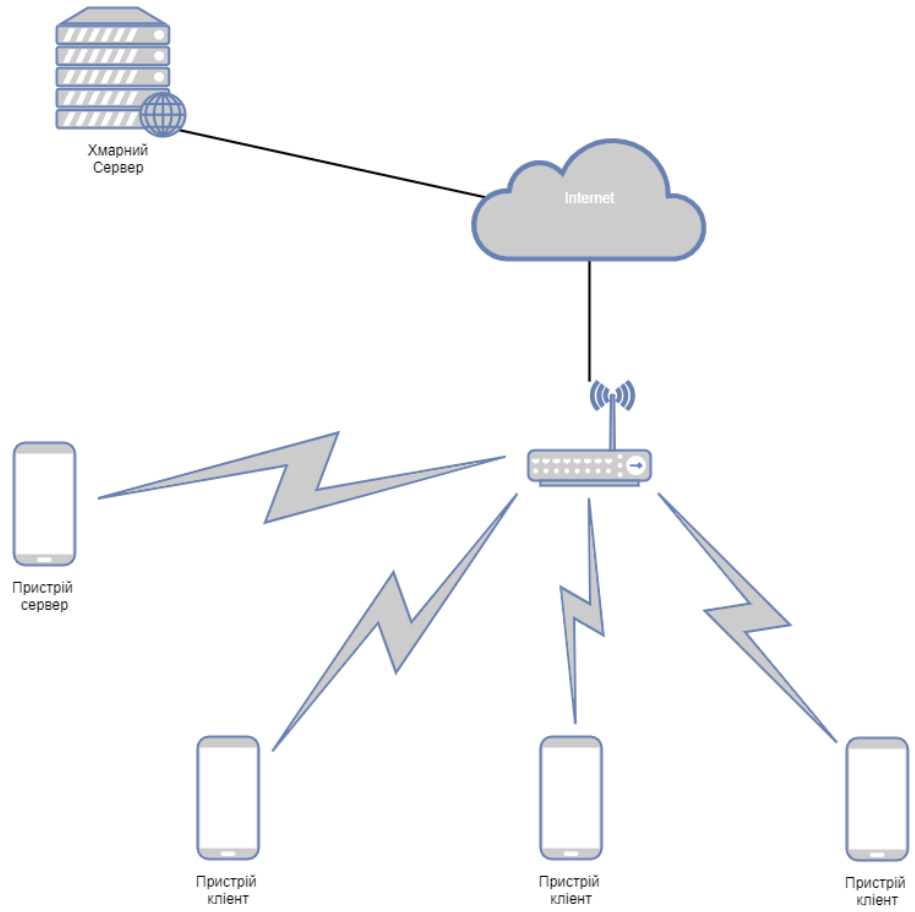

Рисунок 5.8 — Топологія з окремим мережевим маршрутизатором

Топологія з пристроєм сервером у вигляді маршрутизатора потребує наявного режиму «access point» в пристрої організатора заходу, що дозволяє імітувати безпровідну локальну мережу. В такому випадку всі гості під’єднуються до створеної локальної мережі, і вже передача даних відбувається не через Wi-Fi маршрутизатор, а напряму до пристрою сервера. Це дозволяє пришвидшити час передачі даних, але потребує наявного режиму у пристрої. Проте такий підхід дозволяє проводити заходи у віддалених від населених пунктів місцях.

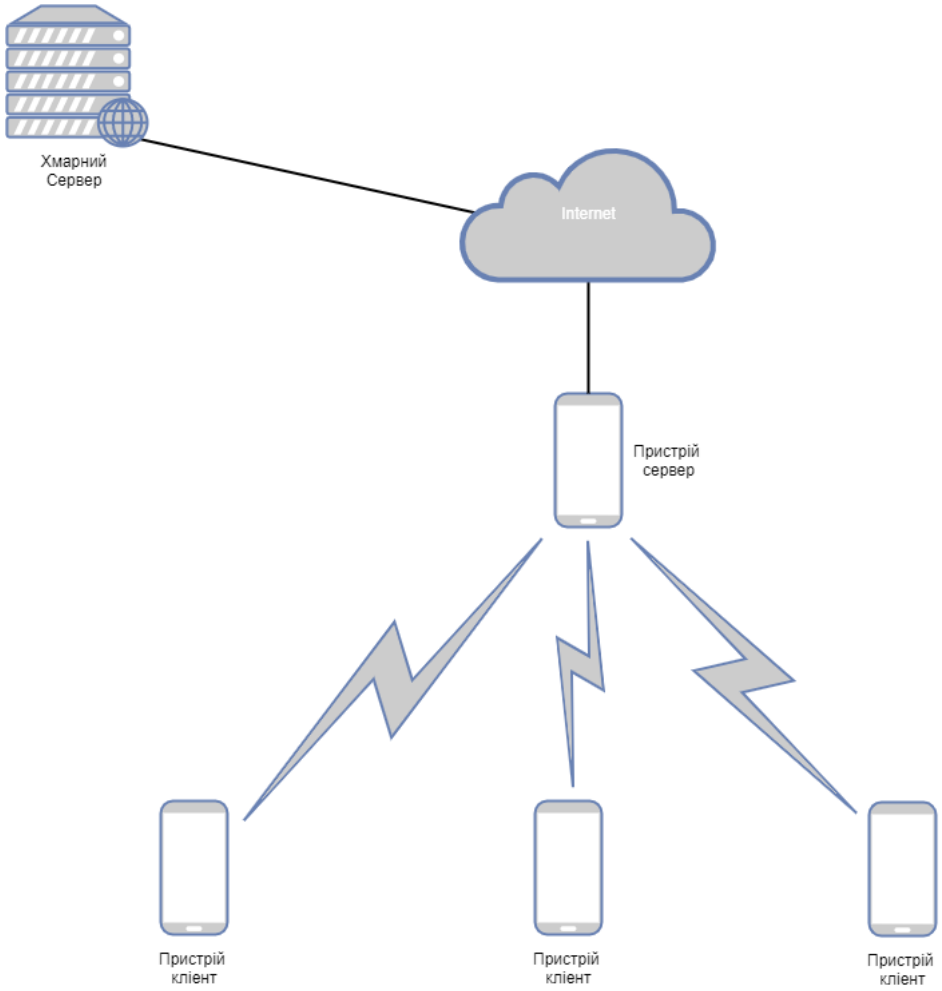

Рисунок 5.9 — Топологія з пристроєм сервером у вигляді маршрутизатора

5.8 Діаграма використання застосунку

Діаграма використання ІА61.080БАК.005 Д2 була створена для представлення взаємодії користувача з мобільним застосунком. Вона допомагає краще зрозуміти

цілі програмного забезпечення, відразу охопити всі можливості системи, і облегшує подальше знайомство з інтерфейсом застосунку.

Так зліва розташований користувач, ним може бути будь-яка людина з пристроєм на який встановлений мобільний застосунок. А в прямокутнику, який являє собою область роботи застосунку, розташовані всі доступні йому функції. На схемі можна побачити перші два блоки використання: «Створити користувача» і «Увійти». Вони описують загальноприйняті можливості створення облікового запису в системі, і містить в собі функції, які виконуються системою під час проходження цієї операції. Додатковою можливістю виступає редагування імені користувача. «Відмітити користувачів» - є основною функцією застосунку, і потребує від користувача обов'язкових дій, таких як «Задати параметри заходу» та «Почати процедуру відмічання», і на додачу має необов'язкову функцію — «Видалити відвідувача». Іншою основною функцією виступає «Відміититись», що потребує вибору необхідного заходу, і допускає можливість користувача покинути той самий захід. Функція «Переглянути історію» є комплексною, і містить в собі перегляд історії для створених та відвіданих заходів.

Даний тип діаграм надає уявлення про всю функціональність застосунку. Відображає всіх учасників, які можуть взаємодіяти з системою. Також узагальнює різні процеси застосунку під одним цілим.

5.9 Діаграма послідовності роботи системи

На діаграмі ІА61.080БАК.005 Д3 представлена послідовність роботи системи. Вона слугує для зображення послідовності певних операцій в мобільному застосунку або її відсутності. Ця діаграма значно облегшує розуміння роботи системи у випадках розділеності її між декількома середовищами. А оскільки мобільний застосунок взаємодіє одночасно з декількома іншими пристроями, і одночасно з хмарним сервісом, ця діаграма є досить необхідною. Також

демонструє основні етапи роботи застосунку, на яких відбувається спілкування між середовищами.

На діаграмі представлений процес від створення заходу і до його завершення. Так першим етапом послідовності виступає отримання персонального ідентифікатора користувача, що вимагає автентифікації. Подальші кроки демонструють спілкування між двома пристроями. Отже, надсилання UDP пакету з інформацією про захід є безперервним процесом впродовж всього існування заходу, а обмін TCP пакетами слугує для відмічання користувача у заході. Так від гостя поступають персональні дані разом з інформацією про адрес пристрою в мережі, а назад відправляється результат відмічання. При неуспішному відмічанні надсилається інформація, яка містить дані про помилку. Наступний умовний блок видалення гостя демонструє послідовність необхідних операцій для його видалення, що потребують як взаємодій напряму з пристроєм користувача та хмарним сервісом. Процес завершення заходу демонструється відправленням збережених даних на сервер.

5.10 Діаграма діяльності застосунку

На діаграмі ІА61.080БАК.005 Д4 представлена діаграма діяльності. Цей вид діаграм є різновидом блок-схем, що представляє алгоритм виклику операцій. На ній представлені дії які відбуваються між можливими варіантами використання застосунку, які представлені на діаграмі ІА61.080БАК.005 Д1.

Діаграма представлена декількома головними елементами. Прямокутник з заокругленнями відображає операції. Ромби — це умовні розгалужувачі. Широкі смуги слугують для розділення та об’єднання зав’язків. А чорне коло та обведене чорне коло символізують початок та кінець задачі.

Після початкового елементу діаграми присутні дії для виконання авторизації, що описують вхід в наявний обліковий запис, створення нового та підтвердження пошти. Процес «Відобразити головне меню» слугує головним місцем

розгалуження загальної роботи застосунку. Процес «Створення заходу» описує основні кроки для успішного початку заходу, і описує дії для скасування або збереження. Проходження процесу відмічання у заході демонструє кроки які виконуються для оновлення списку доступних заходів, та описує алгоритм у випадках успішного і безуспішного відмічання користувача. Перегляд історії демонструє процеси перегляду обраних даних. Кожне розгалуження з перерахованих вище у своїй кінцевій точці повертається до процесу «Відобразити головне меню». Останньою дією для завершення роботи застосунку слугує «Вихід з облікового запису», після якого програма повертається в стан з якого починався алгоритм.

Дана діаграма діяльності слугує для опису динамічної поведінки системи. Представлення можливих послідовностей допомагає краще моделювати поведінку системи без безпосередньої роботи з програмою. Вона допомагає в розробці програмних рішень прямим і реверсивним методами, перший з яких був використаний під час моделювання мобільного застосунку. І також облегшує розуміння поведінкових особливостей програми, разом з розмежуванням застосунку на підзадачі за допомогою представлення умовних точок входу до кожної з них.

6 ТЕСТУВАННЯ СИСТЕМИ

Тестування системи відбувається після її розробки. Воно допомагає виявити помилки в реалізації застосування. Також допомагає впевнитись, що мобільний застосунок перебуває в працездатному стані. Для цього використовують два підходи: модульні тести, та ручне тестування. Перший підхід призначений для пришвидшення процесу отримання стану програми, другий — для виявлення помилок, які неможливо виявити через модульні тести, по причині неможливості, або складності їх написання.

6.1 Ручне тестування.

Під час ручного тестування здійснюється пошук помилок в роботі програмного забезпечення. Цей процес супроводжується моделюванням різних сценаріїв дій користувача. Цим тестувальник перевіряє правильність поведінки системи та працездатність мобільного застосунку, імітуючи звичайні дії користувача системи. Перед початком ручного тестування зазвичай готується план тестування. В ньому описуються найважливіші частини системи для перевірки.

В нашому випадку більша частина тестування буде виконуватись в ручному режимі, оскільки програмний підхід достатньо складно тестується модульними тестами. Наведений нижче плани тестування дозволить перевірити деякі місця програми на недоліки.

Тестування обробки введених даних:

— відсутність деяких даних;

— некоректні дані;

— неіснуючі дані.

Наведені кроки перевіряються на сторінках:

— сторінка входу;

— сторінка реєстрації;

— створення заходу.

Тестування користувацького інтерфейсу відносно доступності мережі:

— доступ до локальної мережі Wi-Fi відсутній;

— доступ лише до локальної Wi-Fi мережі;

— доступ до мережі Internet через Wi-Fi.

Наведені кроки перевіряються на сторінках:

— створення заходу;

— сторінка відмічання.

Проведемо тестування обробки введених даних.

6.1.1 Сторінка входу

На даному вікні присутні 2 поля: адреса електронної пошти і пароль користувача. Вхід не повинен виконуватися при пустих полях, а сама кнопка входу повинна бути неактивна. При такому сценарії будуть відображатися помилки біля текстових полів. Результат сценарію представлений на рисунку 6.1.

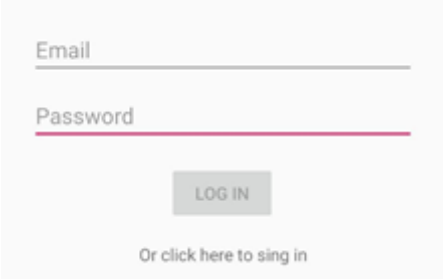

Рисунок 6.1 — Вхід з пустими полями.

Також на вікні присутнє поле для введення електронної пошти, яке підписане «Email», має особливий формат. Цей формат вимагає наявності спец-символу «@», перед яким вказується почтова адреса, а після доменне ім’я сервера, на якому розташована поштова скринька. При введені неправильної пошти відображається відповідне повідомлення, яке представлене на рисунку 6.2.

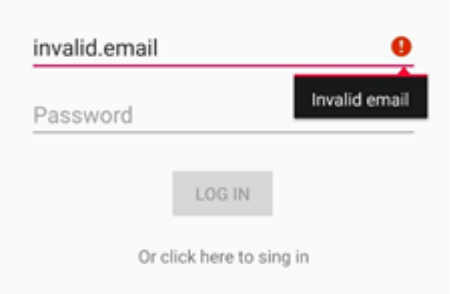

Рисунок 6.2 — Перевірка правильності адреси.

Наступний етап полягає у перевірці застосунку на спробу входу з даними, які відповідають жодній з облікових записів. Результат входу представлений на рисунку 6.3.

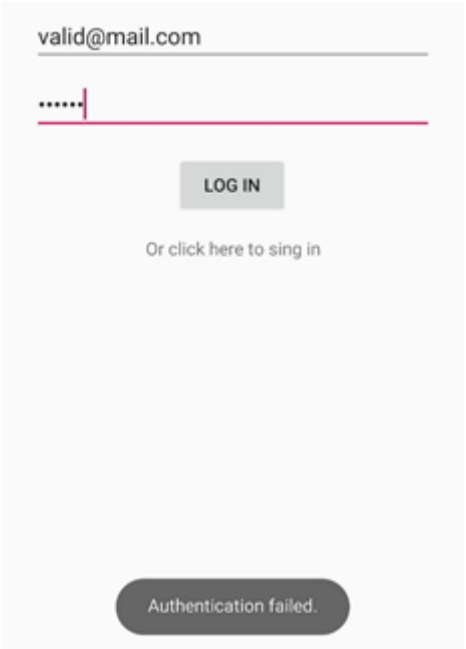

Рисунок 6.3 — Вхід за неіснуючими даними.

За допомогою даного тестування ми пересвідчились в коректній обробці різних типів даних на сторінці входу. Відповідне інформування користувача присутнє.

6.1.2 Сторінка реєстрації

На даному вікні присутні три поля: адреса електронної пошти, пароль та ім’я користувача. Ці поля також є обов’язковими і мають схожі повідомлення. Перевірка реєстрації з неправильними даними буде зайвою, оскільки система

створить нового користувача при неправильних даних. Для спрощення тестування можна перевірити кілька сценаріїв за один крок. Таким чином ми протестуємо перевірку правильності формату електронної пошти та перевірку довжини паролю і імені. Результат перевірки представлений на рисунку 6.4.

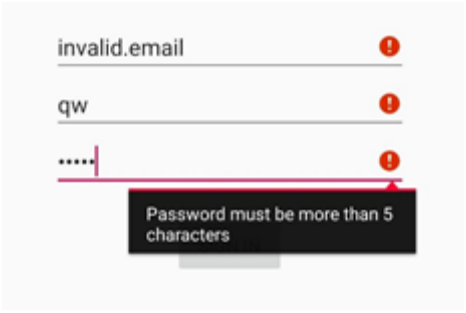

Рисунок 6.4 — Реєстрація з введеними некоректними даними.

Тепер ми впевнились, що користувач створив свій обліковий запис з правильною поштою і паролем. І при наступному вході в систему його дані будуть відповідати критеріям.

Проведемо тестування користувацького інтерфейсу відносно доступності мережі.

6.1.3 Обробка даних на сторінці створення заходу.

На сторінці знаходяться 2 поля: одне для назви, інше для опису. Обов'язковим є лише назва заходу. Поле для опису заповнюється за бажанням. На цій сторінці ми можемо лише перевірити довжину назви, яка повинна перевищувати 2 символи. Результат перевірки представлений на рисунку 6.5.

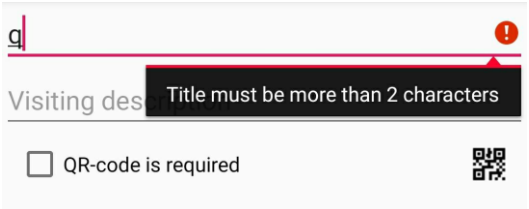

Рисунок 6.5 — Створення заходу з недозволеною довжиною назви.

6.1.4 Доступність мережі на сторінці створення заходу.

На сторінці присутня кнопка старту проходження заходу. Вона повинна при відсутньому підключені до мережі сповіщати користувача про цю необхідність. Результат старту проводження заходу без підключення до мережі представлений на рисунку 6.6.

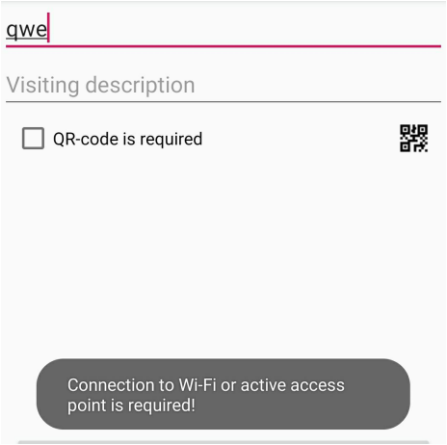

Рисунок 6.6 — Старт проходження заходу без підключення до мережі.

При присутньому підключені до мережі, після початку проведення заходу, ця кнопка повинна перейти в стан завершення цього процесу. А при наявному доступі до глобальної мережі Internet повинна бути доступна кнопка видалення гостя з заходу.

6.1.5 Доступність мережі на сторінці відмічання в заході.

На сторінці відмічання в заході присутній список доступних заходів. При відсутньому підключені до мережі повинен бути присутній текст з інформацією про цю необхідність. Результат старту проводження заходу без підключення до мережі представлений на рисунку 6.9.

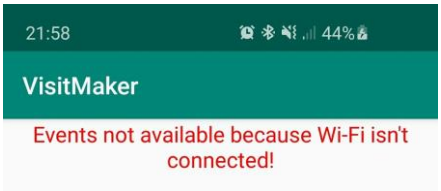

Рисунок 6.9 — Відмічання у заході без підключення до мережі.

Як можна побачити на рисунку 6.9 підключення до Wi-Fi відсутнє і відповідне повідомлення відображається.

При присутньому підключені до мережі, після початку проходження заходу, цей список повинен відобразити доступний захід, або повідомити про відсутність їх наявності. Вікно відмічання в заході ніяк не залежить від підключення до мережі Internet. Тому дане тестування не буде проведене.

Провівши такий вид тестування можна гарантувати, що пристрій знаходиться в задовільному для роботи застосунку стані, оскільки програма сама не може його контролювати, і інтерфейс є зрозумілим для користувача.

6.2 Модульне тестування.

Модульні тести виступають спеціалізованими блоками коду. Їх задача автоматично протестувати окрему функціональність повних класів застосунку. Зазвичай обираються всі публічні методи відповідних класів, і формується спеціальна таблиця результатів, які очікує отримати тестувальник. Вони виконуються незалежно один від одного і дозволяють швидко отримувати результат. Для зручності дані тести групуються у класи, які направлені на тестування певної функціональності.

Для модульного тестування слугують спеціальні бібліотеки. Саме вони запускають написані блоки коду, і перевіряти отриманий результат. Також вони облегшують формування таблиць з очікуваним результатом. Дозволять легко формувати статистику, та вимірюють швидкість проходження тестів. Для тестування мобільного застосунку був обраний фреймворк JUnit.

Виконаємо модульне тестування адаптерів даних для кожної сторінки мобільного застосунку. Дані тести будуть перевіряти правильність форматування даних моделей для представлення їх на користувацькому інтерфейсі. Результат тестування представлено на рисунку 6.10.

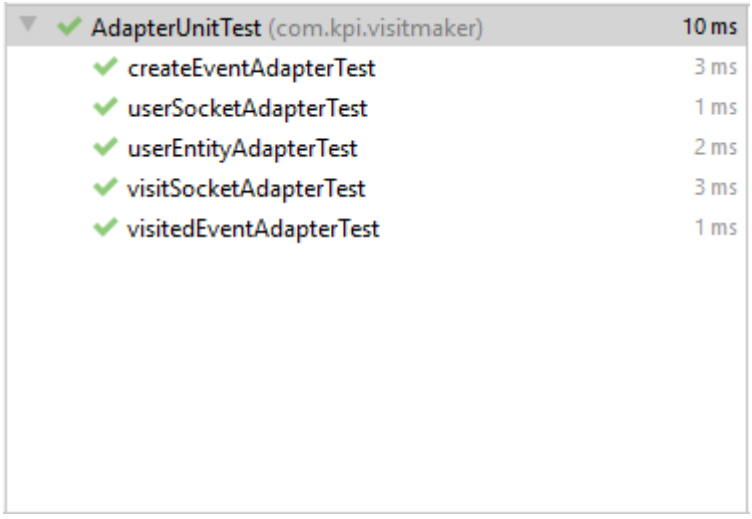

Рисунок 6.10 — Модульне тестування адаптерів.

Успішність пройдених тестів демонструють зелені позначки біля назви кожного тесту. Цей набір модульних тестів продемонстрував правильність роботи адаптерів для контролю коректності відображення елементів на сторінках.

Також буде протестований метод роботи з користувачем, будуть перевірені головні методи роботи з ним. Результат тестування представлено на рисунку 6.11.

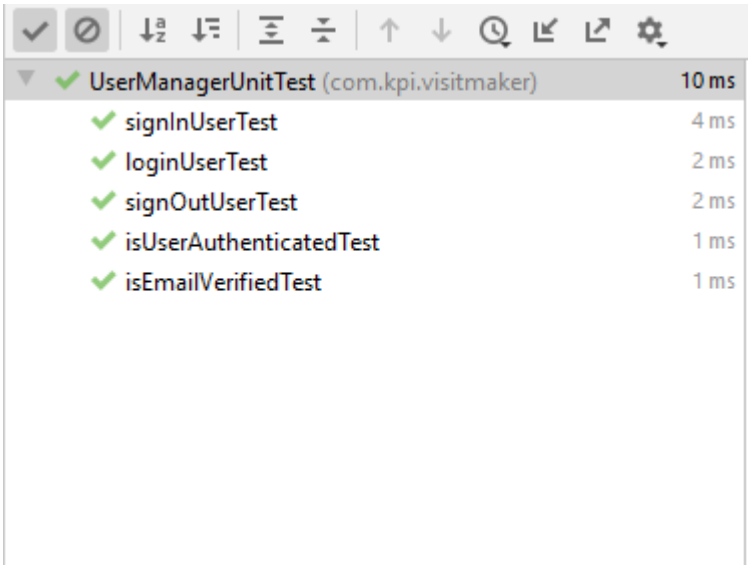

Рисунок 6.11 — Модульне тестування класу роботи з користувачем.

На рисунку 6.11 теж відображаються зелені позначки, які свідчать про успішність тестів. Їх успішність свідчить про те, що клас роботи з обліковим записом користувача функціонує правильно. Перші три тести демонструють правильність роботи методів входу до системи, реєстрації та виходу з облікового заходу користувача. Останні два пересвідчують нас в тому, що в локально в системі збережений поточний обліковий запис та чи пошта користувача підтверджується вірно.

Провівши дане тестування було підтверджено правильність роботи компонентів програми. Проводячи таке тестування періодично можна дуже швидко і якісно перевіряти в якому стані знаходиться мобільний застосунок. Цей підхід економить час розробника та повідомляє його про наявні помилки. В подальшому періодичному тестування можна буде генерувати таблиці статистики тестів, та проводити їх аналіз. Тому можна зробити висновок, що модульне тестування є дуже корисним під час розробки мобільного застосунку.

7 КЕРІВНИЦТВО КОРИСТУВАЧА

7.1 Вхід в систему

Для нових користувачів робота в системі можлива лише після реєстрації в ній. Для цього необхідно на сторінці входу натиснути на текст «Click here to sign in». Після цього потрібно ввести необхідні персональні дані у відповідні поля. В поле для пошти можливо ввести лише текст, який відповідає шаблону пошти, наприклад «example@email.com». Поле для імені користувача потребую більше ніж 2 літери, а поле для пароля — більше ніж 5 літер. Після цього на введену пошту буде надісланий лист, в якому буде посилання, перейшовши по якому ви підтвердите, що саме ви володієте цією поштою. А доти вхід до головного меню буде блокувати вікно, яке повідомляє на яку пошту був відправлений лист, і надає можливість повторно надіслати його. І після підтвердження пошти можна повертатися до мобільного застосунку, де вже після натискання кнопки входу буде наданий доступ до всіх функцій.

Користувачі з наявним обліковим записом можуть увійти в систему через вікно входу. Достатньо ввести пошту та пароль. Поля мають такі самі вимоги що і при реєстрації. І у випадку, якщо користувач вже підтвердив свою пошту, то після натиснення кнопки «Sign In», він потрапить до головного меню.

7.2 Створення заходу

Для створення заходу необхідно в головному меню програми натиснути на кнопку «Init Visiting». Далі слід ввести назву заходу, яка є обов’язковою. Назва заходу повинна мати понад два символи. Поле «Description» є необов’язковим і може бути ігнороване. Якщо користувач потребує більшої надійності, то він може активувати опцію «QR-code is required». Після її активації, для відмічання в заході всім гостям необхідно буде відсканувати цей QR-код. Для відображення QR-коду достатньо натиснути відповідну кнопку з цього відображення, справа від активації

кнопки. Наступним кроком виступає натиснення кнопки «Start Check In», після чого буде розпочате відмічання в заході. Захід не буде розпочатий у разі відсутнього підключення до локальної мережі або активного режиму «access point». І по мірі відмічання гостей, їх імена будуть додаватися в списку по центру екрану. По бажанню власник заходу може видалити небажаного гостя, але для більшої успішності цієї операції бажано мати підключення до Internet. І для завершення створеного заходу слід натиснути на ту ж саму кнопку яка вже матиме надпис «End Check In».

7.3 Відмічання у заході

При натисканні кнопки «Check In» відкривається вікно відмічання у доступних заходах. При цьому користувачу слід переконатися, що він під’єднаний до локальної мережі, і саме в ній проводиться збори. Всі доступні заходи будуть представлені на цьому екрані. Біля кожного буде наявна кнопка з текстом «Check In» для відмічання в ньому, після її натискання користувач з’явиться в списку гостей заходу. Якщо вхід до заходу обмежений скануванням QR-коду, то після натискання кнопки «Check In» відкриється вікно зчитування QR-коду, який ви можете знайти у організатора. Користувач матиме змогу добровільно покинути захід, натиснувши ту ж саму кнопку, але вже з текстом «Leave».

7.4 Додаткові функції застосунку

Очевидно, що застосунок має вікно для перегляду історії, і воно відкривається по натисканні кнопки «History». У цьому вікні доступні два розділи: відвідувані заходи та створені заходи. Перший демонструє список відвіданих заходів, і надає необхідну інформацію у вигляді назви, опису, власника заходу, дати та часу проведення, при цьому список відвідувачів не надається. Другий демонструє список створених користувачем заходів, де вказується ті ж дані, що і для першого

розділу, лише без імені власника, оскільки власником є поточний користувач, і доступний список гостей, який відкривається після натискання трикутника під датою проведення. Можливість видалення гостей зі спис створених заходів не надається, оскільки всі наявні в тому списку могли потрапити туди тільки при явній присутності, і видалення буде нераціональною можливістю.

Доступне редагування імені користувача. Це можна зробити, натиснувши кнопку опцій на головному меню, і обравши пункт «Edit name». Після цього з’явиться вікно з поточним іменем користувача, і відредагувавши його достатньо натиснути кнопку «Save» для його збереження.

Серед додаткових опцій також доступна кнопка виходу з поточного облікового запису. Можливість видалення облікового запису не надається, по причині використанні даних користувача у відвіданих ним заходах.

ВИСНОВКИ

Під час виконання дипломного проєкту був спроєктований, розроблений та протестований мобільний застосунок. Реалізація заточена під операційну систему Android. Система служить для пришвидшення процесу відмічання гостей та направлена на поліпшення зручності проведення заходів у місцях з відсутнім доступом до мережі Internet. І розробка велась під аудиторію, яка представлена організаторами невеликих заходів, керівниками зборів, викладачами, які хочуть ідентифікувати своїх відвідувачів.

Під потреби застосунку були розроблені необхідні діаграми: використання, класів, діяльності, послідовності. До кожної з них був сформований опис, який надає детальну інформацію по реалізації, та обґрунтовує цілі їх створення з подальшими висновками. Ці діаграми розкривають необхідність раніше обраних технологій, і демонструють подробиці роботи компонентів та їх складових, що допомагає оцінити систему загалом.

Сформований опис предметної області дав усвідомити всі архітектурні особливості систем контролю відповідно до обраного типу. Були визначені переваги сучасних мережевих систем контролю, та які інноваційні рішення вони внесли до розвитку даної предметної області. Узагальнено мету автентифікації та перераховано можливі її методи.

Проведений аналіз готових рішень показав переваги і недоліки рішень цієї області. Це дозволило визначити основні направлення, на які буде націлений мобільний застосунок та ідентифікувати проблеми які він буде вирішувати. А сформована загальна таблиця порівняння слугувала орієнтиром для ключових можливостей системи. Окрім того, було визначені передові технології, які використовують готові рішення, що допомогло орієнтуватись в подальшому виборі технологій, і отримати конкурентноспроможний мобільний застосунок на сучасному ринку. Але як і будь-яка система дане рішення не є ідеальним, і в ньому наявні недоліки, на усунення яких мусить бути направлена подальша розробка.

Були сформовані вимоги згідно з поставленими задачами. Відповідно до них поставлені конкретні цілі для розробки, що супроводжувались розбиттям задач на підзадачі. Був сформований план тестування відповідності застосунку до поставлених цілей. Були обґрунтовані причини цих цілей та відповідну користь для системи.

Розвинуті навички розробки мобільних застосунків. Також були здобуті архітектурні здібності під операційну систему Android. Ці навички закріпилися аналізом передових та інноваційних технологій. Таким чином мною були освоєні навички розробки мобільних застосунків мовою Java з використанням сучасного хмарного сервісу Firebase. Поглибивши свої знання в комп'ютеризованих мережевих системах, вдалося забезпечити надійне спілкування через локальну безпровідну мережу між двома мобільними пристроями. Кожна з технологій була детально вивчена для максимальної ефективності системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Система контролю і управління доступом [Електронний ресурс] — Режим доступу: https://pidru4niki.com/12560607/menedzhment/golovni_sistemi_kontrolyu
2. Автентифікація [Електронний ресурс] — Режим доступу: https://auth0.com/docs/application-auth/current
3. OnArrival By Cvent [Електронний ресурс] — Режим доступу: https://www.cvent.com/en/onsite-solutions/onarrival-event-check-in-software
4. Social Tables Check-in [Електронний ресурс] — Режим доступу: https://www.socialtables.com/product/check-in/
5. Booset [Електронний ресурс] — Режим доступу: https://www.boomset.com/
6. Certain Arrive [Електронний ресурс] — Режим доступу: https://www.certain.com/products/arrive/
7. Zkipster [Електронний ресурс] — Режим доступу: https://www.zkipster.com/
8. Клієнт-серверна архітектура [Електронний ресурс] — Режим доступу: https://www.w3schools.in/what-is-client-server-architecture/
9. Багатопоточність [Електронний ресурс] — Режим доступу: https://www.geeksforgeeks.org/multithreading-in-operating-system/
10. Багатозадачність [Електронний ресурс] — Режим доступу: https://medium.com/@rmsrn.85/multitasking-operating-system-types-and-its-benefits-deb1211c1643
11. Firebase [Електронний ресурс] — Режим доступу: https://firebase.google.com/
12. Протокол автентифікації OAuth [Електронний ресурс] — Режим доступу: https://oauth.net/2/
13. Push-сповіщення [Електронний ресурс] — Режим доступу: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications

14. Мова програмування Java [Електронний ресурс] — Режим доступу: https://www.oracle.com/java/

15. Bytecode [Електронний ресурс] — Режим доступу: https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html

16. JVM [Електронний ресурс] — Режим доступу: https://docs.oracle.com/javase/8/docs/technotes/guides/vm/

17. Програмний інтерфейс [Електронний ресурс] — Режим доступу: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

18. Штрих-код [Електронний ресурс] — Режим доступу: https://www.barcodesinc.com/articles/what-is-a-barcode.htm

19. Схема автентифікації через Firebase [Електронний ресурс] — Режим доступу: https://blog.maddevs.io/проверка-истечения-токена-в-firebase-авторизации-проясняем-некоторые-моменты-9e30efa03f8a

ДОДАТОК А

Вихідний код класу «UserManager»

```
public class UserManager {

private FirebaseAuth mAuth = FirebaseAuth.getInstance();

private static UserManager instance;

public static UserManager getInstance() {
    if (instance == null) {
        instance = new UserManager();
    }
    return instance;
}

private UserManager() {
}

public Task<AuthResult> login(String email, String password) {
    return mAuth.signInWithEmailAndPassword(email, password);
}

public Task<AuthResult> signIn(String email, String password) {
    return mAuth.createUserWithEmailAndPassword(email, password);
}

public FirebaseUser getCurrentUser() {
    return mAuth.getCurrentUser();
}

public boolean isUserEmailVerified() {
    FirebaseUser user = mAuth.getCurrentUser();
    if (user != null) {
        return user.isEmailVerified();
    }
    return false;
```

```
    }

    public boolean userIsAuthenticated() {
        return mAuth.getCurrentUser() != null;
    }

    public void reloadUser() {
        FirebaseUser user = mAuth.getCurrentUser();
        if (user != null) {
            user.reload();
        }
    }

    public void signOut() {
        mAuth.signOut();
    }

}
```

ДОДАТОК Б

Вихідний код класу «DiscoverySocketThread»

```
public class DiscoverySocketThread<T extends Serializable> implements Runnable {
private AtomicBoolean running = new AtomicBoolean(false);
private DatagramSocket socket;
private Thread worker;
private T data;

public DiscoverySocketThread(T data) {
    this.data = data;
}

public void startDiscovering() {
    running.set(true);
    worker = new Thread(this);
    worker.start();
}

public void interrupt() {
    running.set(false);
    if (worker != null) {
        closeVisiting();
        worker.interrupt();
    }
}

public boolean isRunning() {
    return running.get();
}

@Override
public void run() {
    try {
        socket = new DatagramSocket();
        socket.setBroadcast(true);

        SocketCommand<T> addCommand = new AddNewSocketCommand<>(data);

        while (running.get()) {
```

```
                List<InetAddress> addresses = listAllBroadcastAddresses();
                for (InetAddress address : addresses) {
                    broadcast(addCommand, address, 8888);
                }

            }
        } catch (SocketException e) {
            if (!socket.isClosed()) {
                Logger.getLogger(DiscoverySocketThread.class.getName()).log(Level.SEVERE,
null, e);
            }
        }


          catch (IOException ex) {
            Logger.getLogger(DiscoverySocketThread.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }


    private void broadcast(byte[] byteArray, InetAddress address, int port) throws
IOException {
        DatagramPacket packet
                = new DatagramPacket(byteArray, byteArray.length, address, port);
        socket.send(packet);
    }

    private void broadcast(Object broadcastObject, InetAddress address, int port) throws
IOException {

        byte[] recvBuf = new byte[15000];

        ByteArrayOutputStream byteStream = new ByteArrayOutputStream(recvBuf.length);
        ObjectOutputStream os = new ObjectOutputStream(new
BufferedOutputStream(byteStream));
        os.flush();
        os.writeObject(broadcastObject);
        os.flush();
        broadcast(byteStream.toByteArray(), address, port);
    }
```

```
    private List<InetAddress> listAllBroadcastAddresses() throws SocketException {
        List<InetAddress> broadcastList = new ArrayList<>();
        Enumeration<NetworkInterface> interfaces
                = NetworkInterface.getNetworkInterfaces();
        while (interfaces.hasMoreElements()) {
            NetworkInterface networkInterface = interfaces.nextElement();

            if (networkInterface.isLoopback() || !networkInterface.isUp()) {
                continue;
            }

            for (InterfaceAddress interfaceAddress :
networkInterface.getInterfaceAddresses()) {
                InetAddress broadcast = interfaceAddress.getBroadcast();
                if (broadcast == null) {
                    continue;
                }

                broadcastList.add(broadcast);
            }
        }
        return broadcastList;
    }

    private void closeVisiting() {
        Runnable closer = new Runnable() {
            @Override
            public void run() {
                try {
                    for (int i = 0; i < 10; i++) {


    final SocketCommand<T> removeCommand = new RemoveSocketCommand<>(data);

                        List<InetAddress> addresses = listAllBroadcastAddresses();
                        for (InetAddress address : addresses) {
                            broadcast(removeCommand, address, 8888);
                        }
                    }
                    socket.close();
```

```
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        };
        Thread closerThread = new Thread(closer);
        closerThread.start();
    }
}
```

ДОДАТОК В

Вихідний код класу «SearchSocketThread»

```
public class SearchSocketThread<T extends Serializable> implements Runnable{

private AtomicBoolean running = new AtomicBoolean(false);
private DatagramSocket socket;
private Thread worker;
private UniqueListAdapter<T> storageAdapter;

public SearchSocketThread(UniqueListAdapter<T> storageAdapter) {
    this.storageAdapter = storageAdapter;
}

public void startSearching() {
    running.set(true);
    worker = new Thread(this);
    worker.start();
}

public void interrupt() {
    running.set(false);
    if (worker != null) {
        worker.interrupt();
    }
    if (socket != null) {
        socket.close();
    }
}

public boolean isRunning() {
    return running.get();
}

@Override
public void run() {
    try {
        //Open a random port to send the package
        socket = new DatagramSocket(8888, InetAddress.getByName("0.0.0.0"));
        socket.setBroadcast(true);
```

```
            while (running.get()) {

                //Wait for a response
                byte[] recvBuf = new byte[15000];
                DatagramPacket receivePacket = new DatagramPacket(recvBuf,
recvBuf.length);
                socket.receive(receivePacket);

                ByteArrayInputStream byteStream = new ByteArrayInputStream(recvBuf);
                ObjectInputStream is = new ObjectInputStream(new
BufferedInputStream(byteStream));
                Object receivedObject = is.readObject();
                try {
                    final SocketCommand<T> socketCommand = (SocketCommand)
receivedObject;

                        if (socketCommand instanceof RemoveSocketCommand) {
                        System.out.println("Remove command");
                    } else {
                        System.out.println("Add command");
                    }
                    new Handler(Looper.getMainLooper()).post(new Runnable() {
                        @Override
                        public void run() {
                            if (socketCommand.updateList(storageAdapter.getItems())) {
                                storageAdapter.notifyDataSetChanged();
                            }
                        }
                    });
                } catch (ClassCastException e) {
                    e.printStackTrace();
                }
            }
        } catch (IOException | ClassNotFoundException ex) {
            Logger.getLogger(CheckInActivity.class.getName()).log(Level.SEVERE, null,
ex);
        }
```

ДОДАТОК Г

Вихідний код класу «SocketClientTask»

```
public class SocketClientTask<T extends Serializable> implements Callable<Boolean> {

private SocketCommand<T> command;
private InetAddress ip;
private int port;
private Thread worker;
private FutureTask<Boolean> future;

public SocketClientTask(InetAddress ipAddress, int port, SocketCommand<T> command) {
    this.command = command;
    this.ip = ipAddress;
    this.port = port;
}

public void startTask() {
    future = new FutureTask<>(this);
    worker = new Thread(future);
    worker.start();
}

public boolean get() {
    boolean result = false;
    try {
        result = future.get();
    } catch (ExecutionException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    return result;
}

public boolean isDone() {
    return future.isDone();
}

@Override
public Boolean call() throws Exception {
```

```
        try {
            Socket clientSocket = new Socket(ip, port);

ObjectOutputStream os = new ObjectOutputStream(clientSocket.getOutputStream());
            os.flush();
            os.writeObject(command);
            os.flush();
    ObjectInputStream is = new ObjectInputStream(clientSocket.getInputStream());
            Object response = is.readObject();
            boolean result = false;
            if (response.equals(CommandResult.SUCCESS)) {
                result = true;
            }
            clientSocket.close();
            return result;
        } catch (IOException e) {
            System.out.println("Failed check in: " + e.getMessage());
            return false;
        }
    }
}
```

ДОДАТОК Д

Вихідний код класу «SocketServerTread»

```
public class SocketServerThread<T extends Serializable> implements Runnable {

    private static final int SocketServerPORT = 8080;

    private AtomicBoolean running = new AtomicBoolean(false);

    private ServerSocket serverSocket;
    private Thread worker;
    private UniqueListAdapter<T> storageAdapter;

    private InetAddress address;
    private int port;

    public SocketServerThread(InetAddress address, int port, UniqueListAdapter<T>
storageAdapter) {
        this.address = address;
        this.port = port;
        this.storageAdapter = storageAdapter;
    }

    public void startServer() {
        running.set(true);
        worker = new Thread(this);
        worker.start();
    }

    public void interrupt() {
        running.set(false);
        worker.interrupt();
        try {
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public boolean isRunning() {
        return running.get();
```

```
}

@Override
public void run() {
    try {
        serverSocket = new ServerSocket(port, 30, address);

        while (running.get()) {
            Socket socket = serverSocket.accept();
            AcceptSocketThread acceptSocketThread =  new AcceptSocketThread(socket);
            acceptSocketThread.start();

        }


      } catch (IOException e) {
        e.printStackTrace();
    }
}

private class AcceptSocketThread extends Thread {

    private Socket socket;

    public AcceptSocketThread(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            ObjectInputStream is = new ObjectInputStream(socket.getInputStream());
            Object receivedObject = is.readObject();


            final SocketCommand<T> socketCommand = (SocketCommand) receivedObject;
            if (socketCommand instanceof RemoveSocketCommand) {
                System.out.println("Remove command");
            }
            new Handler(Looper.getMainLooper()).post(new Runnable() {
                @Override
```

```
                    public void run() {
                        if (socketCommand.updateList(storageAdapter.getItems())) {
                            storageAdapter.notifyDataSetChanged();
                        }
                    }
                });

                ObjectOutputStream os = new ObjectOutputStream(socket.getOutputStream());
                os.flush();
                os.writeObject(CommandResult.SUCCESS);
                os.flush();
                socket.close();

            } catch (IOException | ClassNotFoundException | ClassCastException e) {
                e.printStackTrace();
            }
        }
    }

}
```